EPFL

Thèse n° 8992

Protecting privacy through metadata analysis

Présentée le 21 octobre 2022

Faculté informatique et communications
Laboratoire d'ingénierie de sécurité et privacy
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Sandra Deepthy SIBY

Acceptée sur proposition du jury

Prof. M. J. Payer, président du jury
Prof. C. González Troncoso, directrice de thèse
Prof. A. Markopoulou, rapporteuse
Dr-Ing. B. Stock, rapporteur
Prof. K. Argyraki, rapporteuse

École
polytechnique
fédérale
de Lausanne

2022

Sandra: I don't know whether to be paranoid and check everything again.

Narseo: 20 min before deadline, you won't fix anything critical. And if you find something fixable in that short time, it will be very minor. So? :)

— Narseo's excellent advice for pre-submission panic (and life)

To family...

Acknowledgements

As clichéd as the saying may be, it did take a village to raise this PhD. Below, I will attempt to convey my appreciation for several people who have supported me these last five years.

First and foremost, I would like to thank my advisor, Prof. Carmela Troncoso. Carmela has been instrumental in my growth as a researcher; she has taught me how to ask the right questions, how to best design experiments to answer those questions, how to zoom out and look at the big picture, and how to express my work coherently. Her curiosity, enthusiasm, and determination to achieve excellence in whatever she does have been very inspirational for me. Carmela is a unicorn advisor – she gives me autonomy but is always there when I am stuck, she doles out excellent advice (and memes), and most importantly, she is a great human being. She has raised the bar for my expectations of any person I will work with in the future.

I would like to thank my committee members: Prof. Katerina Argyraki, Prof. Athina Markopoulou, and Dr.-Ing Ben Stock, for taking the time to review my dissertation and for the insightful questions during my defense, and Prof. Mathias Payer, for presiding over my defense. Thank you also to Prof. Jean-Pierre Hubaux and Prof. James Larus for reviewing the introduction of my dissertation, and Holly Cogliati-Bauereis for her editorial suggestions.

I was fortunate to work with a wonderful set of collaborators over the course of my PhD; this dissertation would not have been possible without them: Marc Juarez, Narseo Vallina-Rodriguez, Claudia Diaz, Umar Iqbal, Shaoor Munir, Steven Englehardt, Zubair Shafiq, Ludovic Barman, Christopher Wood, Marwan Fayed, and Nick Sullivan. It has been an absolute joy working with these brilliant (and nice) researchers from whom I've learned a lot. Special thanks to Zubair for his mentorship and for helping me navigate the second half of my thesis; Narseo for his wisdom and kindness (and for giving me the quote for my dissertation); Umar and Marc for always being ready to chat and for giving me great advice; Steve for being a patient mentor during my internship. I would also like to thank all the semester project students who have worked with me; discussions with them have been enlightening and have influenced many of the experiments in my work. I am grateful to Luke Crouch, my manager during my internship, for giving me an

opportunity to explore the area of web tracking. Thank you to Nils Ole Tippenhauer, for having introduced me to security/privacy research and for teaching me so many skills that came in handy during the PhD.
Working and interacting with the (past/present) members of the SPRING lab has been a great experience, and I will miss TA-ing/lunches/presentation feedback sessions/fun discussions/ranting about paper reviews/joking about blockchain with this lovely bunch of people. In particular, I would like to thank: Laurent, for tirelessly assisting me in setting up experiments and improving my code, for reviewing the French abstract of this dissertation, and for his (occasionally, dark) sense of humor; Isabelle, for helping me with all the administrative tasks; Wouter, for patiently shepherding us clueless PhD students; Kasra, for always being up for a coffee together and for his never-ending optimism; Theresa, for showing me that bar charts can be beautiful and for being the nicest office-mate; Bogdan, for giving me hair-envy and for not just being a great colleague but also a great friend. I'm also thankful to the members of our sister labs – LDS, DEDIS, and HexHive; while the pandemic sadly reduced our interactions to a large extent, I fondly remember our many corridor conversations.
Being a student representative has been a big part of my PhD life. I would like to thank my co-representative, Ehsan, for being a great teammate; the past/present EDIC committee (especially Ola/Mathias/Eileen/Cecilia/Annalisa/Marta) for always taking the time to listen to and respond to our complaints (or 'suggestions'); EPIC and the other student representatives for their passion and active participation in improving student life at EPFL. In spite of frustrations at the slow pace of change, these people have inspired me to keep on fighting the good fight.
I am also lucky to have a fantastic group of friends who have made me feel at home in Lausanne. I have thoroughly enjoyed coffees/dinners/various outings/lying on the couch and doing nothing/random discussions with them. Special thanks to the following people for having been a big part of my life these last five years: Bernd, for his sense of humor and for intimidating us all with his biking achievements; George, for his 'I-don't-knows' and for almost being a part of the furniture in our house; Karen, for the amazing food, for entertaining us with her hole-digging, and for being a rock; Mariam, for the interesting conversations and for being habibti; Sena, for all the fun chats and for being my 'Turkish daughter'. And thank you to many others, including Delio, Jakab, Nadia, Pavithra, Sahand, and Sriniketh, for the nice times we have had together. I would also like to thank my friends from Singapore (the NUS and the SUTD crowd); despite being scattered across the globe these days, conversations with them make me feel like we are together.
My family has played a huge role in my well-being during my PhD. I would like to thank my mother-in-law, Dr. Saradhamani, my sister-in-law, Bhargavi, and my nephew, Vivaan, for their constant encouragement and support. They have created great memories during my holidays which helped me get a much-needed break from time to time. I am extremely

grateful to my parents, Dr. Siby and Dr. Deepthy, for their unconditional love, for teaching me to face challenges without fear, and for encouraging me to aim for the stars. My father, the doctor (with the heart of a computer engineer), kicked off my journey in computer science by bringing 'a strange machine' home; he is always there for me with his words-of-wisdom and his calm strength. My mother, probably the only person who follows my Google Scholar profile (and who diligently read up on DNS after my paper), bolsters me with her energy and her great advice; she still the person I want to call first when I achieve something. I want to thank my brother and the (self-proclaimed) 'person who made my sister less boring', Sachin, for being my biggest cheerleader and for always being up for a nonsensical (or serious) conversation. His drive, curiosity, and willingness to venture into the unknown always motivate me. Finally, huge thanks to my husband and long-time partner-in-crime (and now, strictest cricket coach), Bharath. He has patiently dealt with the ups-and-downs of my PhD life, pushed me to go out of my comfort zone, and never failed to make me laugh. This journey would have been a lot less fun without his love, endless conversations, and terrible puns.

Lausanne, September 3, 2022 Sandra Deepthy Siby

Abstract

Although encryption hides the content of communications from third parties, *metadata*, i.e., the information attached to the content (such as the size or timing of communication) can be a rich source of details and context. In this dissertation, we demonstrate the power of metadata analysis. We illustrate ways in which we can use metadata analysis to protect privacy, by addressing two problems in the areas of network and web privacy.

In the first problem, we study recently standardized protocols such as encrypted DNS and QUIC. We show how metadata analysis can be used by adversaries to perform website-fingerprinting attacks against these protocols and to infer websites visited by a user, thereby compromising their privacy. We use the insights from our analysis to identify the requirements for developing effective countermeasures and to improve the resistance of these protocols to website fingerprinting. We find that hiding metadata is challenging.

In the second problem, we address the issue of online advertising and tracking services (ATS) that are constantly evolving to evade privacy protections established by browser vendors. We demonstrate how the very fact of metadata being hard to hide can be useful to defenders. Defenders can use metadata analysis to build ATS-detection systems that are more robust against adversarial evasion by capturing behavioral metadata of ATS. We use our findings to detect and counter the emergence of tracking via first-party cookies.

Keywords: privacy, metadata, website fingerprinting, network protocols, online advertising and tracking, cookies.

Résumé

Bien que le chiffrement cache le contenu des communications aux tiers, les métadonnées, c'est-à-dire les informations attachées au contenu (telles que la taille ou le moment de la communication) peuvent être une source riche de détails et de contexte. Dans cette thèse, nous démontrons la puissance de l'analyse des métadonnées. Nous illustrons des manières dont nous pouvons utiliser l'analyse des métadonnées pour protéger la vie privée, en abordant deux problèmes dans les domaines de la confidentialité des réseaux et du Web.

Dans le premier problème, nous étudions des protocoles récemment normalisés tels que le DNS chiffré et QUIC. Nous montrons comment l'analyse des métadonnées peut être utilisée par des adversaires pour réaliser des attaques par empreintes numériques contre ces protocoles et pour déduire quels sites Web ont été visités par un utilisateur, compromettant ainsi sa vie privée. Nous utilisons les résultats de notre analyse pour identifier les conditions nécessaires au développement de contre-mesures efficaces et pour améliorer la résistance de ces protocoles à l'analyse de l'empreinte numérique. Nous constatons que le masquage des métadonnées est difficile.

Dans le second problème, nous abordons la question des services de publicité et de tracking en ligne (ATS) qui évoluent constamment pour échapper aux protections de la vie privée établies par les fournisseurs de navigateurs. Nous démontrons comment le fait même que les métadonnées soient difficiles à cacher peut être utile aux défenseurs. Les défenseurs peuvent utiliser l'analyse des métadonnées pour construire des systèmes de détection des ATS plus robustes contre l'évasion adverse en capturant les métadonnées du comportement des ATS. Nous utilisons nos découvertes pour étudier et détecter l'émergence du tracking par le biais de cookies de première partie.

Mots clés : vie privée, métadonnées, empreintes de sites Web, protocoles de réseau, publicité et suivi en ligne, cookies.

Contents

1 Introduction

With the spread of digital services, people are now more connected than ever, and have access to unprecedented levels of information and communication. Although this increased connectivity has accelerated social, technological, and economic activity, it also has its negative aspects in the form of increasing threats to user security and privacy. For example, users can be tracked [1], surveilled [2, 3], censored [4, 5], and profiled by various parties in the digital ecosystem.

These risks have prompted Internet governance, industry actors, and standardization bodies to foster privacy protections [6, 7]. There has been an increase in adoption and standardization of protocols for encrypting communication to provide protection against eavesdroppers. However, even though encryption hides the content of communications, an interested party can glean a significant quantity of details from the communication *metadata*. Metadata refers to the *data about the data*. For example, the metadata that characterizes the access of a webpage over an encrypted channel would include details such as the timestamps of the user requests, the size of the traffic, and the end-points of the communication. In contrast, the content of such a communication, rendered invisible to a network adversary due to encryption, would be the HTTPS requests and responses sent when the webpage is accessed. Even if the metadata does not reveal the content of the communication, it can provide additional context that can be helpful for a party monitoring the communication. For example, if the adversary observes that a user connects to `taylorswiftfans.com` on multiple occasions and sends a large number of packets to this end-point, they could conclude that the user is interested in Taylor Swift even without access to the content of the communication. We note that metadata is not restricted to network traffic as shown in the example; one can find metadata associated with files, geolocation data, web pages, and even in non-digital contexts such as museum artifacts.

The purpose of this dissertation is to illustrate ways in which we can use metadata analysis to protect user privacy. We first show metadata analysis from the point of view of an *adversary* who uses it to compromise the network privacy of users. We attempt to develop defenses based on the insights we gain from this analysis, and find that hiding metadata is a difficult problem. We then switch roles and demonstrate how the very fact of metadata being hard to hide can be used by a *defender* to protect the web privacy of users.

Contributions

This dissertation is divided into two parts - one that addresses network privacy, and the other that addresses web privacy. In the first part, we explore how metadata analysis can be used to understand the susceptibility of networking protocols to traffic analysis. Our work provides insights into how to improve the resistance of protocols against website fingerprinting attacks. In the second part, we study how metadata analysis can be used to build more robust detection systems against online advertising and tracking services (ATS). Our research provides promising directions for browser vendors to follow when they develop ATS blocking tools.

Part 1: Metadata Analysis for Network Privacy

Website fingerprinting is a traffic analysis technique that enables an adversary eavesdropping on the network to infer which web pages are visited by a user over an encrypted channel. Encryption of the channel implies that the adversary does not have authorized access to the content of the network traffic. Hence, in website fingerprinting, the adversary uses network traffic metadata (such as packet sizes and timing) to identify web pages. In a typical website fingerprinting pipeline, an adversary collects a dataset of network traces, trains a classifier on metadata features extracted from these traces, and uses this classifier to classify network traffic from their targeted user.

Most prior work in website fingerprinting focuses on Tor traffic. In this dissertation, we investigate whether recently standardized protocols, such as encrypted DNS [8] [9] and QUIC [10], are susceptible to website fingerprinting, and if so, whether we can develop effective defenses against them.

In Chapter 2, we describe our study of website fingerprinting on encrypted DNS protocols (DNS-over-HTTPS and DNS-over-TLS). In Chapter 3, we turn our attention to defenses against website fingerprinting, by trying to develop a defense for QUIC traffic. We describe the two chapters in detail below.

Chapter 2: Encrypted DNS $\Longrightarrow$ Privacy? A Traffic Analysis Perspective.

Today, the bulk of DNS queries are sent in the clear, a situation that can be exploited to track users online, to deploy DNS-based censorship, and to engage in mass surveillance. These problems prompted the standardization of two protocols for protecting user privacy: DNS-over-TLS (DoT) [8] and DNS-over-HTTPS (DoH) [9]. Both DoT and DoH encrypt the communication between the client and the recursive DNS resolver to prevent the inspection of the requested domain names by network eavesdroppers. In 2018, Google and Cloudflare launched public DoH resolvers [11, 12], and in 2020, DoH became the default DNS protocol for US users who used Firefox [13]. Evaluations of existing DoH implementations have focused on understanding the effect of encryption on performance and cost, but have not investigated whether they achieve their purported goal of protecting privacy.

In Chapter 2, we try to fill the gap by investigating the protection that encrypted DNS provides against website fingerprinting. We propose a novel feature set to perform the attacks, as features used to attack HTTPS or Tor traffic are not suitable for DNS. Our work shows that the domains resolved using DNS can be recovered from metadata, and that metadata is sufficient to enable DNS-based censorship, thus effectively defeating the protection offered by encryption. More concretely, our work demonstrates that the analysis of network metadata enables an adversary to identify the web pages visited by a user with high accuracy, and enables a censor to selectively drop DNS connections in order to implement censorship. This attack uses 124 times fewer data than traffic analysis on HTTPS flows, which ensures the scalability of these attacks. In addition to showing the feasibility of the attack in laboratory conditions, we evaluate real-world settings in our study, identifying the settings where the attack is most accurate in practice. We find that factors, such as location, resolver, platform, and client, do alter attack performances, but they are not yet able to completely stop them.

We also investigate possible defenses. We show that the padding schemes proposed in the specifications of the encrypted DNS protocols, in particular, are not effective or they impose a large overhead in the communication. In the case of DNS traffic, only aggressive measures, such as using Tor, are a good defense.

Chapter 3: Help Needed! Understanding QUIC `PADDING`-Based defenses against Website Fingerprinting.

QUIC is a connection-oriented protocol built on top of UDP that aims to provide low-latency, multiplexed, secure communication with less head-of-line blocking and faster connection migration. QUIC was standardized in May 2021 and is currently being developed by the IETF [10]. Prior work has shown that QUIC is susceptible to website

fingerprinting [14]. The IETF QUIC specification describes a QUIC PADDING frame [15] as a frame with no semantic value that can be used to increase packets size and to provide protection against traffic analysis.

In Chapter 3, we study whether we can build effective website fingerprinting defenses at the network layer by using only the PADDING frame, as envisioned by the standard. We confirm previous claims that network-layer padding cannot provide good protection against powerful adversaries that are capable of observing all traffic traces. In contrast to prior work, we also demonstrate that such padding is ineffective, even against adversaries with constraints on traffic views and processing power. Network-layer padding without application input is ineffective because it fails to hide information unique across different applications, such as the total size of a website. Such information can be gleaned only from the application layer. Therefore, we explore whether this information can be effectively collected at the application layer and whether it can be used to inform padding algorithms in an effective manner. We find that websites currently utilize a vast amount of resources that are served, in a significant number of cases, by third parties. We demonstrate that if all parties do not participate in defending against website fingerprinting, an adversary can successfully identify the page and, as a result, leave users vulnerable. We identify current web development practices that hinder the deployment of effective website fingerprinting defenses. We show that, unless these practices change, protecting users is close to impossible, as it would require significant bandwidth overhead and coordination among parties under very dynamic conditions. We provide recommendations for guiding future efforts in the development of effective defenses against website fingerprinting.

Part 2: Metadata Analysis for Web Privacy

Users rely on privacy-enhancing blocking tools to protect themselves from online advertising and tracking. Many of these tools (e.g., uBlock Origin, Ghostery, Firefox, Brave, etc.) rely on manually curated filter lists to block advertising and tracking. Yet, manual curation has scalability issues: filter lists cannot keep up with the rapidly evolving ATS ecosystem. In order to tackle this, the research community has been developing machine-learning (ML) approaches in order to automate the detection of ATS, and to make filter lists more comprehensive.

Although the state-of-the art ML approaches effectively detect ATS, thereby solving the scalability issues, little is known about their robustness against adversarial evasion by ATS. Therefore, in Chapter 4, we study the robustness of ML-based approaches to ATS blocking, and we build, WebGraph, our system that uses ATS-behavior metadata and is significantly more robust than state-of-the-art approaches. In Chapter 5, we explore the use of first-party cookies for tracking, and we build CookieGraph, a system that

uses the lessons learned from Chapter 4 to classify first-party ATS cookies. We describe the two chapters in detail below.

Chapter 4: WebGraph: Capturing Advertising and Tracking Information Flows for Robust Blocking.

The earlier ML approaches for blocking detect ATS, either at the network and JavaScript layers of the web stack. To extract features and train ML models to detect ATS, network layer approaches rely on content present in URLs, HTTP headers, and request and response payloads (e.g., keywords, query strings, payload size). Whereas, JavaScript layer approaches rely on static or dynamic analysis to extract features and train ML models. Newer approaches reason that ATS use all three layers of the web stack (i.e., network, JavaScript, and HTML) for their execution, and hence gather information from all the layers instead of a single layer. These approaches are graph-based, i.e., they build a graph representation of events that occur during a web page load, and they train ML models on features extracted from the graph representation.

In Chapter 4, we study the robustness of the graph-based approaches against adversarial evasion by third parties, i.e., where a third party aims to get a classifier to mis-classify its ATS resources as benign. We find that current graph-based approaches have an over-reliance on easy-to-manipulate content features (information related to individual nodes in the graph, such as URL length and presence of ad/tracking keywords in the URL). We introduce WebGraph, the first ML-based approach that detects ATS based on their actions rather than their content. By featuring the actions that are fundamental to advertising-and-tracking information flows – e.g., storing an identifier in the browser or sharing an identifier with another tracker – WebGraph performs almost as well as prior approaches, but is significantly more robust to adversarial evasions. We show that WebGraph is effective in reducing the success rate, not only of adversaries that perform content-based attacks but also of adversaries that employ sophisticated evasion techniques that are beyond those currently deployed on the web.

Chapter 5: CookieGraph: Measuring and Countering First-Party Tracking Cookies.

Browser vendors and ATS are engaged in an escalating arms race. As soon as browsers deploy privacy protections, e.g., third-party cookie blocking, ATS quickly adapt to evade them, e.g., CNAME-cloaking, bounce tracking. To gain an edge over browser vendors, ATS are increasingly exploiting browser features that are typically used for functional purposes and hence cannot be trivially blocked. One prominent technique is the use of first-party context to store ATS cookies. Prior research has measured the deployment

of first-party cookies by ATS, and has surmised that these might be used in cross-site tracking.

In Chapter 5, we study how first-party cookies are used for cross-site tracking. First, to understand the scale of first-party cookie usage in tracking, we conduct a large-scale measurement study of websites, with and without third-party cookie blocking. Our study enables us to not only identify typical characteristics of ATS first-party cookies but also shows that there is a rise in the use of such tracking techniques in response to countermeasures deployed by browser vendors. Using the first-party ATS cookie behaviors that we identified from the measurement study, we build CookieGraph, an ML-based classifier for robust mitigation of first-party cookie-based tracking. We show that CookieGraph can identify first-party ATS cookies with high accuracy, and can be used in blocking-solutions with minimal breakage (loss of legitimate functionality).

Summary

In summary, the contributions outlined in this dissertation are as follows:

- We demonstrate that the protections offered by recently standardized encrypted DNS protocols can be thwarted by metadata analysis to identify websites visited by users.
- We find that padding-based countermeasures against website fingerprinting, as specified in the QUIC standard, are ineffective. We identify the challenges that have to be addressed in order to deploy an effective website fingerprinting defense.
- We show that current approaches to ATS detection are not robust against adversarial evasion. We develop WebGraph, a system that relies on metadata of ATS behavior to accurately and robustly identify ATS for blocking.
- We discover an increased use of first-party cookies by ATS for cross-site tracking in response to browser privacy protections. To address this, we develop CookieGraph, which uses ATS behavior metadata to target these first-party cookies.

Bibilographic notes

The contents of this dissertation are based on the following:

Chapter 2: S. Siby, M. Juarez, N. Vallina-Rodriguez, C. Troncoso, "DNS Privacy not so private: the traffic analysis perspective", in *Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETS)*, 2018.

Chapter 2: S. Siby, M. Juarez, C. Diaz, N. Vallina-Rodriguez, C. Troncoso, "Encrypted DNS $\Longrightarrow$ Privacy? A Traffic Analysis Perspective.", in *Network and Distributed System Security Symposium (NDSS)*, 2020.

Chapter 3: S. Siby, L. Barman, C. A. Wood, M. Fayed, N.Sullivan, C. Troncoso, "Help Needed! Understanding QUIC `PADDING`-based defenses against website fingerprinting.", *to be submitted*, 2022.

Chapter 4: S. Siby, U. Iqbal, S. Englehardt, Z. Shafiq, C. Troncoso, "WebGraph: Capturing Advertising and Tracking Information Flows for Robust Blocking.", in *USENIX Security*, 2022.

Chapter 5: S. Munir, S. Siby, U. Iqbal, S. Englehardt, C. Troncoso, Z. Shafiq, "CookieGraph: Measuring and Countering First-Party Tracking Cookies.", *under submission*, 2022.

Part I

Metadata analysis for network privacy

2 Encrypted DNS $\Longrightarrow$ Privacy? A Traffic Analysis Perspective

This chapter is based on the articles:

"DNS Privacy not so private: the traffic analysis perspective", S. Siby, M. Juarez, N. Vallina-Rodriguez, C. Troncoso, in *Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETS)*, 2018.

"Encrypted DNS $\Longrightarrow$ Privacy? A Traffic Analysis Perspective.", S. Siby, M. Juarez, C. Diaz, N. Vallina-Rodriguez, C. Troncoso, in *Network and Distributed System Security Symposium (NDSS)*, 2020.

2.1 Introduction

Regular Domain Name System (DNS) requests have been mostly sent in the clear [6]. This situation enables entities, such as Internet Service Providers (ISPs), Autonomous Systems (ASes), or state-level agencies, to perform user tracking, mass surveillance [2, 3] and censorship [4, 5]. The risk of pervasive surveillance and its consequences has prompted Internet governance, industry actors, and standardization bodies to foster privacy protections [6, 7]. In particular, for DNS these bodies have standardized two protocols: DNS-over-TLS (DoT) [8] and DNS-over-HTTPS (DoH) [9]. These protocols encrypt the communication between the client and the recursive resolver to prevent the inspection of domain names by network eavesdroppers. These standardization bodies also consider protection mechanisms to limit inference of private information from traffic metadata, such as the timing and size of network packets of the encrypted DNS communication. These mechanisms protect against traffic analysis by padding traffic [16], or by multiplexing the encrypted DNS traffic with other traffic, *e.g.*, when DoH and web HTTPS traffic share a single TLS tunnel (see §8.1 [9]).

During 2018, Google and Cloudflare launched public DoH resolvers [11, 12], while Mozilla added DoH support to Firefox [17] and has been gradually rolling it out as the default configuration for Firefox users in the US since September 2019 [13]. These efforts aim to leverage DoH and DoT's capacity to enhance users' browser traffic security guarantees [18]. Yet, it is known that even when communications are encrypted, traffic features such as volume and timing can reveal information about their content [19, 20, 21, 22, 23, 24]. As of today, existing evaluations of DoH implementations have focused on understanding the impact of encryption and transport protocols on performance [25, 26], and on cost [27, 28].

In this chapter, we aim to fill this gap by studying the effectiveness of traffic analysis attacks in revealing users' browsing patterns from encrypted DNS. We focus our analysis on DoH, as its adoption by large industry actors (e.g., Google, Cloudflare, and Mozilla) makes it prevalent in the wild. For completeness, we include a comparison of the protection provided by Cloudflare's DoH and DoT resolvers. In our analysis, we consider a passive adversary, as described in RFC 7626 [6], who is placed between the client and the DNS resolver. The adversary's goal is to identify which web pages users visit, to either perform surveillance or censorship. As the RFC stresses, this adversary may be in "a different path than the communication between the initiator [*e.g.*, the client] and the recipient [*e.g.*, a web server]" [6], and thus can launch attacks even if they do not see all the traffic between endpoints.

We find that features traditionally used in atttacks on web traffic [19, 20, 21, 22, 29, 30] are not suitable for encrypted DNS traffic. This is because DNS, as opposed to web traffic, is bursty, chatty, and is mostly composed of small packets. We engineer a *novel set of features* that focuses on local traffic features, and enables successful identification of requested websites on encrypted DNS. As encrypted DNS traces are much smaller that their web traffic counterparts, our techniques require *124 times less data than state-of-the-art traffic analysis on web traffic*, allowing adversaries to run attacks at large scale [31]. Furthermore, our new feature set on encrypted DNS traffic is *as effective or more so* than state-of-the-art attacks on web traffic in identifying web pages.

We also find that differences between the environment used by the adversary to train the attack (e.g., user location, choice of client application, platform or recursive DNS resolver), and the environment where the attack is actually deployed, negatively affect the performance of the attack. Most prior work on traffic analysis assumes the adversary knows the environment where the attack will be deployed, but the adversary cannot trivially obtain that information a priori. Our features allow the adversary to infer that information and thus tailor the attacks accordingly, maintaining high attack performance for each specific environment.

Next, we evaluate existing traffic analysis defenses, including the standardized EDNS0 padding [16] —implemented by Cloudflare and Google in their solutions—, and the use of Tor [32] as transport, a feature available when using Cloudflare's resolver. We find that, unless EDNS0 padding overhead is large, current padding strategies cannot completely prevent our attack. Also, while Tor offers little protection against web page fingerprinting on web traffic [20, 21, 22, 33], Tor is an extremely effective defense against web page fingerprinting on encrypted DNS traffic.

Finally, we measure the potential of encryption to hinder DNS-based censorship practices. We show that with encryption, it is still possible to identify which packet carries the DNS lookup for the first domain. We quantify the collateral damage of blocking the response to this lookup, thereby preventing the user from seeing any content. We also show that, to minimize the effect of censorship on non-blacklisted pages, censors must wait to see, on average, 15% of the encrypted DNS traffic.

Our main contributions are as follows:

- We show that the features for traffic analysis existing in the literature are not effective on encrypted DNS. We propose a new feature set that results in successful attacks on encrypted DNS and that outperforms existing attacks on HTTPS (Section 2.5.1).
- We show that web page fingerprinting on DoH achieves the same accuracy as web page fingerprinting on encrypted web traffic, while requiring *124* times less volume of data. We show that factors such as end-user location, choice of DNS resolver, and client-side application or platform, have a negative impact on the effectiveness of the attacks, but do not prevent them (Section 2.5).
- We evaluate the traffic analysis defenses proposed in the standard and show that they are not effective. We find that in the case of encrypted DNS, contrary to web traffic, routing over Tor deters web page identification on encrypted DNS traffic (Section 2.6).
- We evaluate the feasibility of DNS-based censorship when DNS look-ups are encrypted. We show the censor can identify the packet with the first domain lookup. We quantify the trade-off between how much content from a blacklisted page the user can download, and how many non-blacklisted pages are censored as a side effect of traffic-analysis-based blocking (Section 2.7).
- We gather the first dataset of encrypted DNS traffic collected in a wide range of environments (Section 2.4).[1]

[1]Dataset and code at: https://github.com/spring-epfl/doh_traffic_analysis

Impact Upon responsible disclosure of our attacks, Cloudflare changed their DoH resolver to include padding. This work was also invited to an IETF Privacy Enhancements and Assessments Research Group Meeting, and will contribute to the next RFC for traffic analysis protection of encrypted DNS.

2.2 Background and Related Work

In this section, we provide background on the Domain Name System (DNS) and existing work on DNS privacy.

The Domain Name System (DNS) is primarily used for translating easy-to-read domain names to numerical IP addresses[2]. This translation is known as domain resolution. In order to resolve a domain, a client sends a DNS query to a *recursive resolver*, a server typically provided by the ISP with resolving and caching capabilities. If the domain resolution by a client is not cached by the recursive name server, it contacts a number of *authoritative name servers* which hold a distributed database of domain names to IP mappings. The recursive resolver traverses the hierarchy of authoritative name servers until it obtains an answer for the query, and sends it back to the client. The client can use the resolved IP address to connect to the destination host. Figure 2.1 illustrates this process.

Enhancing DNS Privacy. Security was not a major consideration in the first versions of DNS, and for years DNS traffic was sent in the clear over (untrusted) networks. Over the last few years, security and privacy concerns have fostered the appearance of solutions to make DNS traffic resistant to eavesdropping and tampering. Several studies have empirically demonstrated how the open nature of DNS traffic is being abused for performing censorship [36, 37] and surveillance [38, 2]. Early efforts include protocols such as DNSSEC [39] and DNSCrypt [40]. DNSSEC prevents manipulation of DNS data using digital signatures. It does not, however, provide confidentiality. DNSCrypt, an open-source effort, provides both confidentiality and authenticity. However, due to lack of standardization, it has not achieved wide adoption.

In 2016, the IETF approved DNS-over-TLS (DoT) [8] as a Standards Track protocol. The client establishes a TLS session with a recursive resolver (usually on port TCP:853 as standardized by IANA [8]) and exchanges DNS queries and responses over the encrypted connection. To amortize costs, the TLS session between the client and the recursive DNS resolver is usually kept alive and reused for multiple queries. Queries go through this channel in the same manner as in unencrypted DNS – chatty and in small volume.

In DoH, standardized in 2018, the local DNS resolver establishes an HTTPS connection

[2]Over time, other applications have been built on top of DNS [34, 35]

to the recursive resolver and encodes the DNS queries in the body of HTTP requests. DoH considers the use of HTTP/2's Server Push mechanism. This enables the server to preemptively push DNS responses that are likely to follow a DNS lookup [18], thus reducing communication latency. As opposed to DoT, which uses a dedicated TCP port for DNS traffic and thus is easy to monitor and block, DoH look-ups can be sent along non-DNS traffic using existing HTTPS connections (yet potentially blockable at the IP level).

There are several available implementations of DoT and DoH. Since 2018, Cloudflare and Quad9 provide both DoH and DoT resolvers, Google supports DoH, and Android 10 has native support for DoT. DoH enjoys widespread support from browser vendors. Firefox provides the option of directing DNS traffic to a *trusted recursive resolver* such as a DoH resolver, falling back to plain-text DNS if the resolution over DoH fails. In September 2019, Google announced support for DoH in version 78 of Chrome [41]. Cloudflare also distributes a stand-alone DoH client and, in 2018, they released a hidden resolver that provides DNS over Tor, not only protecting look-ups from eavesdroppers but also providing anonymity for clients towards the resolver. Other protocols, such as DNS-over-DTLS [42], an Experimental RFC proposed by Cisco in 2017, and DNS-over-QUIC [43], proposed to the IETF in 2017 by industry actors, are not widely deployed so far.

Several academic works study privacy issues related to encrypted DNS. Shulman suggests that encryption alone may not be sufficient to protect users [44], but does not provide any experiments that validate this statement. Our results confirm their hypothesis that encrypted DNS response size variations can be a distinguishing feature. Herrmann *et al.* study the potential of DNS traces as identifiers to perform user tracking, but do not consider encryption [45]. Finally, Imana *et al.* study privacy leaks on traffic between recursive and authoritative resolvers [46]. This is not protected by DoH, and it is out of scope of our study.

2.3 Problem Statement

We study if it is possible to infer which websites a user visits by observing encrypted DNS traffic. This information is of interest to multiple actors, *e.g.*, entities computing statistics on Internet usage [47, 48], entities looking to identify malicious activities [49, 50, 5], entities performing surveillance [38, 2], or entities conducting censorship [51, 37].

We consider an adversary that can collect encrypted DNS traffic between the user and the DNS recursive resolver (red dotted lines in Figure 2.1), and thus, can link look-ups to a specific origin IP address. Such an adversary could be present in the users' local network, near the resolver, or anywhere along the path (e.g., an ISP or compromised

network router). As noted in the RFC, this adversary may be "in a different path than the communication between the initiator and the recipient" [6].

This adversary model also includes adversaries that only see DoH traffic, e.g., adversaries located in an AS that lies between the user and the resolver but not between the user and the destination host. In order to confirm that this adversary is possible, we conducted an experiment where we ran traceroutes to a subset of the websites we use in the study (1,445 websites), and to two resolvers – Cloudflare and Google. We intersected the AS sets and observed that the sets do not fully overlap in 93% and 90% of the cases for Cloudflare and Google respectively. Furthermore, we note that BGP hijacking attacks, which are becoming increasingly frequent [52], can be used to selectively intercept paths to DoH resolvers. In such cases, the adversary can only rely on DNS fingerprinting to learn which webpages are visited by a concrete user for monitoring, or censorship [2, 38]. If the adversary is actually in the path to the users' destination, she could perform traditional website fingerprinting. However, we show that web page fingerprinting on only DoH traffic achieves the same accuracy while requiring (on average) 124 times less data than attacking HTTPS traces that include web traffic. This is critical to guarantee the scalability of attacks to a large number of targets [31].

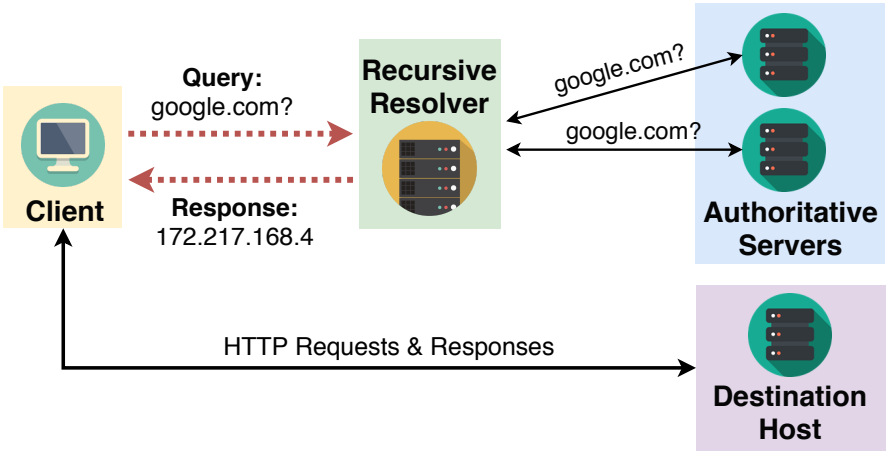


Figure 2.1: DNS resolution: To visit `www.google.com`, a user queries the recursive resolver for its IP. If the record is not cached, the recursive resolver queries an authoritative resolver and forwards the response to the client. The client uses the IP in the response to connect to the server via HTTP. We consider an adversary placed between the client and the resolver (i.e., observes the red dotted lines).

We assume that the adversary has access to *encrypted DNS traffic traces* that are generated when the user visits a website via HTTP/S using DoH to resolve the IPs of the resources. An encrypted DNS trace, which we also call DoH trace, comprises the resolution of the

visited website's first-party domain, and the subsequent resolutions for the resources contained in the website, e.g., images and scripts. For instance, for visiting Reddit, after resolving `www.reddit.com`, the client would resolve domains such as `cdn.taboola.com`, `doubleclick.net` and `thumbs.redditmedia.com`, among others.

We consider two adversarial goals. First, *monitoring* the browsing behavior of users, which we study in Section 2.5; and, second, *censoring* the web pages that users visit, which we address in Section 2.7. These two goals differ in their data collection needs. Monitoring adversaries can collect full traces to make their inferences as accurate as possible, as they do not take any action based on their observations. In contrast, censorship adversaries need to find out which domain is being requested as fast as possible so as to interrupt the communication. Thus, they act on partial traces.

2.4 Data Collection

To collect data we set up Ubuntu 16.04 virtual machines with DoH clients that send DNS queries to a public DoH resolver. We use Selenium[3] (version 3.14.1) to automatically visit a webpage from our list, triggering DNS look-ups for its resources. We restart the browser in between webpage visits to ensure that the cache and profile do not affect collection. We capture network traffic between the DoH client and the resolver using *tcpdump*, and we filter the traffic by destination port and IP to obtain the final DoH traffic trace.

We collect traces for the top, middle, and bottom 500 webpages in Alexa's top-million websites list on 26 March 2018, 1,500 webpages in total. We note that even though 1,500 is still a small world compared to the size of the Web, the largest world considered in evaluations similar to ours for website fingerprinting on web traffic over Tor is of 800 websites [53, 33, 54, 21, 55, 22].

We visit each webpage in a round-robin fashion, obtaining up to 200 samples for every webpage. For our open world analysis, we collect traces of an additional 5,000 webpages from the Alexa top-million list. We collected data during two periods, from 26 August 2018 to 9 November 2018, and from 20 April 2019 to 14 May 2019. Data from these two periods is never mixed in the analysis.

We consider different scenarios varying DoH client and resolver, end user location and platform, and the use of DNS traffic analysis defenses (padding, ad-blocker usage, DNS-over-Tor). Table 2.1 provides an overview of the collected datasets. In order to better understand the vulnerability of DNS encryption to traffic analysis, we designed heterogeneous experiments that look into different aspects of the problem, resulting in multiple

[3] https://www.seleniumhq.org/

datasets of varied sizes and collected under different conditions – in many cases, several datasets for each experiment type. In each experiment, we vary one characteristic (*e.g.*, location or platform), and keep the default configuration for the rest.

We also collected a dataset (using the Stubby client and Cloudflare's resolver) to study the effectiveness of defenses on DoT traffic as compared to DoH. Since Cloudflare had already implemented padding of responses for DoT traffic at the time of data collection, we were not able to collect a dataset of DoT without padding. In the following sections, we use the Identifier provided in the second column to refer to each of the datasets. Note that unless specified otherwise, we use Cloudflare's DoH client.

Table 2.1: Overview of datasets. Our default configuration is a Desktop located in Lausanne where Firefox uses Cloudflare's client to query Cloudflare's resolver. Changes are detailed in the table (see Appendix for a detailed version).

Name	Identifier	Location	# webpages	# samples
Desktop (Location 1)	LOC1	Lausanne	1,500	200
Desktop (Location 2)	LOC2	Leuven	1,500	60
Desktop (Location 3)	LOC3	Singapore	1,500	60
Raspberry Pi	RPI	Lausanne	700	60
Firefox with Google resolver	GOOGLE	Leuven	700	60
Firefox with Cloudflare resolver	CLOUD	Leuven	700	60
Firefox with Cloudflare client	CL-FF	Leuven	700	60
Open World	OW	Lausanne	5,000	3
DoH and web traffic	WEB	Leuven	700	60
DNS over Tor	TOR	Lausanne	700	60
Cloudflare's EDNS0 padding	EDNS0-128	Lausanne	700	60
Recommended EDNS0 padding	EDNS0-468	Lausanne	700	60
EDNS0 padding with ad-blocker	EDNS0-128-adblock	Lausanne	700	60
DoT with Stubby client	DOT	Lausanne	700	60

Data curation. We curate the datasets to ensure that our results are not biased. First, we aim at removing the effect of spurious errors in collection. We define those as changes in the websites for reasons other than those variations due to their organic evolution that do not represent the expected behavior of the page. For instance, pages that go down during the collection period.

Second, we try to eliminate website behavior that is bound to generate classification errors unrelated to the characteristics of DNS traffic. For instance, different domains generating the same exact DNS traces, *e.g.*, when webpages redirect to other pages or to the same resource; or web servers returning standard errors (e.g., 404 Not Found or 403 Forbidden).

For curation, we use the Chrome over Selenium crawler to collect the HTTP request/responses, not the DNS queries responses, of all the pages in our list in LOC1. We conduct

two checks. First, we look at the HTTP response status of the *FQDN (Fully Qualified Domain Name)*, that is being requested by the client. We identify the webpages that do not have an HTTP OK status. These could be caused by a number of factors, such as pages not found (404), anti-bot mechanisms, forbidden responses due to geoblocking [56] (403), internal server errors (500), and so on. We mark these domains as invalid. Second, we confirm that the FQDN is present in the list of requests and responses. This ensures that the page the client is requesting is not redirecting the browser to other URLs. This check triggers some false alarms. For example, a webpage might redirect to a country-specific version (`indeed.com` redirecting to `indeed.fr`, results in `indeed.com` not being present in the list of requests); or in domain redirections (`amazonaws.com` redirecting to `aws.amazon.com`). We do not consider these cases as anomalies. Other cases are full redirections. Examples are malware that redirects browser requests to `google.com`, webpages that redirect to GDPR country restriction notices, or webpages that redirect to domains that specify that the site is closed. We consider these cases as invalid webpages and add them to our list of invalid domains.

We repeat these checks multiple times over our collection period. We find that 70 webpages that had invalid statuses at some point during our crawl, and 16 that showed some fluctuation in their status (from invalid to valid or vice versa). We study the effects of keeping and removing these webpages in Section 2.5.2.

2.5 DNS-based Website Fingerprinting

Website fingerprinting attacks enable a local eavesdropper to determine which web pages a user is accessing over an encrypted or anonyimized channel. It exploits the fact that the size, timing, and order of TLS packets are a reflection of a website's content. As resources are unique to each webpage, the traces identify the web even if traffic is encrypted. Website fingerprinting has been shown to be effective on HTTPS [30, 29, 57], OpenSSH tunnels [58, 59], encrypted web proxies [60, 61] and VPNs [62], and even on anonymous communications systems such as Tor [63, 33, 53, 54, 19, 20, 21, 22].

The patterns exploited by website fingerprinting are correlated with patterns in DNS traffic: which resources are loaded, and their order determines the order of the corresponding DNS queries. Thus, we expect that website fingerprinting can also be effective on DNS encrypted flows such as DNS-over-HTTPS (DoH) or DNS-over-TLS (DoT). We call *DNS fingerprinting* the use of traffic analysis to identify the web page that generated an encrypted DNS trace, i.e., website fingerprinting on encrypted DNS traffic. In the following, whenever we do not explicitly specify whether the target of website fingerprinting is DNS or HTTPS traffic, we refer to traditional website fingerprinting on web traffic over HTTPS.

2.5.1 DNS fingerprinting

As in website fingerprinting, we treat DNS fingerprinting as a supervised learning problem: the adversary first collects a training dataset of encrypted DNS traces for a set of pages. The page (label) corresponding to a DNS trace is known. The adversary extracts features from these traces (e.g., lengths of network packets) and trains a classifier that, given a network trace, identifies the visited page. Under deployment, the adversary collects traffic from a victim and feeds it to the classifier to determine which page the user is visiting.

Traffic variability. Environmental conditions introduce variance in the DNS traces sampled for the same website. Thus, the adversary must collect multiple DNS traces in order to obtain a robust representation of the page.

Some of this variability has similar origin to that of web traffic. For instance, changes on the website that result in varying DNS look-ups associated with third-party embedded resources (e.g., domains associated to ad networks); the platform where the client runs, the configuration of the DoH client, or the software using the client which may vary the DNS requests (e.g., mobile versions of websites, or browsers' use of pre-fetching); and the effects of content localization and personalization that determine which resources are served to the user.

Additionally, there are some variability factors specific to DNS traffic. For instance, the effect of the local resolver, which depending on the state of the cache may or may not launch requests to the authoritative server; or the DNS-level load-balancing and replica selection (e.g., CDNs) which may provide different IPs for a given domain or resource [64].

Feature engineering. Besides the extra traffic variability compared to web traffic, DNS responses are generally smaller and more chatty than web resources [64, 65]. In fact, even when DNS lookups are wrapped in HTTP requests, DNS requests and responses fit in one single TLS record in most cases. These particularities hint that traditional website fingerprinting features, typically based on aggregate metrics of traffic traces such as the total number of packets, total bytes, and their statistics (e.g., average, standard deviation), are inadequate to characterize DoH traffic. We test this hypothesis in the following section, where we show that state-of-the-art web traffic attacks' performance drops in 20% when applied on DoH traces (see Table 2.3).

To address this problem, we introduce a novel set of features, consisting of n-grams of TLS record lengths in a trace. Following the usual convention in website fingerprinting, we represent a traffic trace as a sequence of integers, where the absolute value is the size of

the TLS record and the sign indicates direction: positive for packets from the client to the resolver (outgoing), and negative for the packets from the resolver to the client (incoming). An example of this representation is the trace: $(-64, 88, 33, -33)$. Then, the uni-grams for this trace are $(-64), (88), (33), (-33)$, and the bi-grams are $(-64, 88), (88, 33), (33, -33)$. To create the features, we take tuples of n consecutive TLS record lengths in the DoH traffic traces and count the number of their occurrences in each trace.

In some of our experiments, we used a proxy to man-in-the-middle the DoH connection between the client and the resolver[4], and obtained the OpenSSL TLS session keys using Lekensteyn's scripts[5]. In all the decrypted TLS records we observe only one single DoH message (either a request or a response). However, as Houser *et al.* indicate, some clients and resolvers' implementations result on multiple DoT messages in the same TLS record [66].

The intuition behind our choice of features is that n-grams capture patterns in request-response size pairs, as well as the local order of the packet size sequence. To the best of our knowledge, n-grams have never been considered as features in the website fingerprinting literature.

We extend the n-gram representation to traffic bursts. Bursts are sequences of consecutive packets in the same direction (either incoming or outgoing). Bursts correlate with the number and order of resources embedded in the page. Additionally, they are more robust to small changes in order than individual sizes because they aggregate several records in the same direction. We represent n-grams of bursts by adding lengths of packets in a direction inside the tuple. In the previous example, the burst-length sequence of the trace above is $(-64, 121, -33)$ and the burst bi-grams are $(-64, 121), (121, -33)$.

We experimented with uni-, bi- and tri-grams for both features types. We observed a marginal improvement in the classifier on using tri-grams at a substantial cost on the memory requirements of the classifier. Bi-grams capture the request-response nature of DNS traffic. We also experimented with the timing of packets. As in website fingerprinting [33], we found that it does not provide reliable prediction power. This is because timing greatly varies depending on the state of the network and thus is not a stable feature to fingerprint web pages. In our experiments, we use the concatenation of uni-grams and bi-grams of both TLS record sizes and bursts as feature set.

Algorithm selection. After experimenting with different supervised classification algorithms, we selected Random Forests (RF) which are known to be very effective for traffic analysis tasks [21, 67].

[4]https://github.com/facebookexperimental/doh-proxy
[5]https://git.lekensteyn.nl/peter/wireshark-notes

Random forests (RF) are ensembles of simpler classifiers called decision trees. Decision trees use a tree data structure to represent splits of the data: nodes represent a condition on one of the data features and branches represent decisions based on the evaluation of that condition. In decision trees, feature importance in classification is measured with respect to how well they split samples with respect to the target classes. The more skewed the distribution of samples into classes is, the better the feature discriminates. Thus, a common metric for importance is the Shannon's entropy of this distribution. Decision trees, however, do not generalize well and tend to overfit the training data [68]. RFs mitigate this issue by randomizing the data and features over a large amount of trees, using different subsets of features and data in each tree. The final decision of the RF is an aggregate function on the individual decisions of its trees. In our experiments, we use 100 trees and a majority vote to aggregate them.

Validation. We evaluate the effectiveness of DNS fingerprinting in two scenarios typically used in the website fingerprinting literature. A *closed world*, in which the adversary knows the set of all possible pages users may visit; and an *open-world*, in which the adversary only has access to a set of *monitored* pages, and the user may visit pages outside of this set.

In the closed world, we evaluate the effectiveness of our classifier measuring the per-webpage *Precision*, *Recall* and *F1-Score*. We consider positives as DoH traces generated by that webpage and negatives as traces generated by any other webpage. For each webpage, true positives are DoH traces generated by visits to the webpage that the classifier correctly assigns to that webpage; false positives are traces generated by vists to other pages that are incorrectly classified as the webpage; false negatives are traces of the webpage that are classified as other pages; and true negatives are traces of other pages that are not classified as the webpage. Then, Precision is the ratio of true positives to the total number of traces that were classified as positive (true positives and false positives); Recall is the ratio of true positives to the total number of positives (true positives and false negatives); and the F1-score is the harmonic mean of Precision and Recall. We aggregate these metrics over all the webpages, providing their average and standard deviation.

In the open world there are only two classes: monitored (positive) and unmonitored (negative). Thus, a true positive in the open world is a trace of a monitored webpage that is classified as monitored, and a false positive is a trace of an unmonitored page that is classified as monitored. Likewise, a true negative is a trace of an unmonitored page classified as unmonitored and a false negative a trace of a monitored page classified as unmonitored.

We use 10-fold cross-validation, a standard methodology in machine learning, to measure the generalization error of the classifier, also known as *overfitting*. In cross-validation, the samples of each class are divided in ten disjoint partitions. The classifier is then trained on each set of nine partitions and tested in the remaining one. Since there are $\binom{10}{9} = 10$ possible sets of nine partitions, this provides us ten samples of the classifier performance on a set of samples on which it has not been trained on. Taking the average and standard deviation of these samples gives us an estimate of the performance of the classifier on unseen examples.

2.5.2 Evaluating *n*-grams features

We now evaluate the effectiveness of n-grams features to launch DNS fingerprinting attacks. We also compare these features with traditional website fingerprinting features in both DoH traffic and web traffic over HTTPS.

Evaluation in closed and open worlds. We first evaluate the attack in a closed world using the LOC1 dataset. We try three settings: i. an adversary that attacks the full dataset of 1,500 websites, ii. an adversary that attacks the curated dataset of 1,414 websites after we eliminate spurious errors, and iii. an adversary that attacks the full dataset but considers regional versions of given pages to be equivalent. For example, classifying `google.es` as `google.co.uk`, a common error in our classifier, is considered a true positive. We see in Table 2.2 that testing on the clean dataset offers just a 1% performance increase, and that considering regional versions as equivalent results provides an additional 3% increase. Given the minimal differences, in the remainder of the experiments we use the full dataset.

Table 2.2: Classifier performance for LOC1 dataset (mean and standard deviation for 10-fold cross validation).

Scenario	Precision	Recall	F1-score
Curated traces	0.914 ± 0.002	0.909 ± 0.002	0.908 ± 0.002
Full dataset	0.904 ± 0.003	0.899 ± 0.003	0.898 ± 0.003
Combined labels	0.940 ± 0.003	0.935 ± 0.003	0.934 ± 0.003

In the context of website fingerprinting, Overdorf *et al.* [69] showed that it is likely that the classifier's performance varies significantly between different individual classes. Thus, looking only at average metrics, as in Table 2.2, may give an incomplete and biased view of the classification results. To check if this variance holds on DNS traces we study the performance of the classifier for individual websites. The result is shown in Figure 2.2. In this scatterplot each dot is a website and its color represents the absolute difference between Precision and Recall: blue indicates 0 difference and red indicates maximum difference (i.e., $|Precision - Recall| = 1$). We see that some websites (red dots on the right

of the Precision scatterplot) have high Recall – they are often identified by the classifier, but low Precision – other websites are also identified as the website. Thus, these websites have good privacy since the false positives provide the users with plausible deniability. For other pages (red dots on the right of the Recall scatterplot), the classifier obtains low Recall – it almost never identifies them, but high Precision – if they are identified, the adversary is absolutely sure her guess is correct. The latter case is very relevant for privacy, and in particular, censorship, as it enables the censor to block without fear of collateral damage.

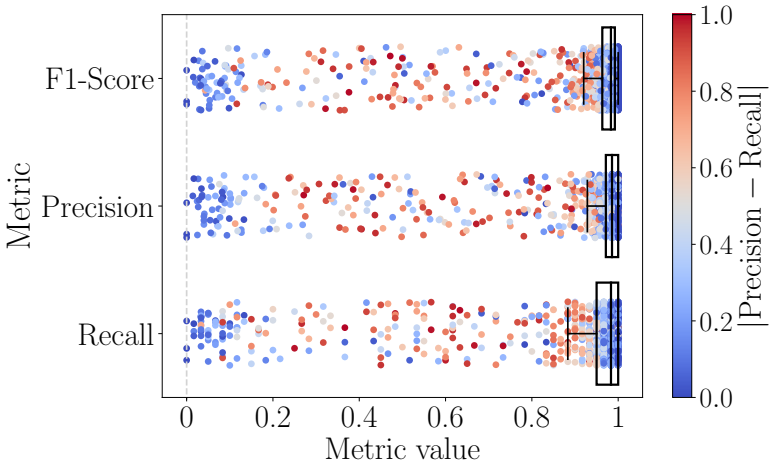


Figure 2.2: Performance per class in LOC1. Each dot represents a class and its color the absolute difference between Precision and Recall (blue low, red high).

Open world. In the previous experiments, the adversary knew that the webpage visited by the victim was within the training dataset. We now evaluate the adversary's capability to distinguish those webpages from other unseen traffic. Following prior work [67, 70] we consider two sets of webpages, one *monitored* and one *unmonitored*. The adversary's goal is to determine whether a test trace belongs to a page within the monitored set.

We train a classifier with both monitored and unmonitored samples. Since it is not realistic to assume that an adversary can have access to all unmonitored classes, we create unmonitored samples using 5,000 webpages traces formed by a mix of the OW and LOC1 datasets. We divide the classes such that 1% of all classes are in the monitored set and 10% of all classes are used for training. We select monitored pages at random. This assumes that different adversaries may be interested in blocking different pages, and enables us to evaluate the average case. We ensure that the training dataset is balanced, i.e., it contains equal number of monitored and unmonitored samples; and the test set contains an equal number of samples from classes used in training and classes unseen by the classifier. When performing cross validation, in every fold, we consider a different

combination of the monitored and unmonitored classes for training and testing so that we do not overfit to a particular case.

To decide whether a target trace is monitored or unmonitored, we use a method proposed by Stolerman *et al.* [71]. We assign the target trace to the monitored class if and only if the classifier predicts this class with probability larger than a threshold t, and to unmonitored otherwise. We calculate the Precision-Recall ROC curve for the monitored class using scikit-learn's precision-recall curve plotting function, which varies the discrimination threshold, t. The curve is shown in Figure 2.3, where the notches indicate the varying t.

We also plot in Figure 2.3 the curve corresponding to a random classifier, a naive classifier that outputs positive with probability the base rate occurrence of the positive class. This classifier serves as baseline to assess the effectiveness of the n-grams. When we vary the discrimination threshold of the random classifier, Precision remains constant, i.e., the threshold affects TPs and FPs in the same proportion. The effect of the threshold in FNs, however, is inversely proportional to TNs. Thus, Recall changes depending on the classifier's threshold. The AUC (Area Under Curve) for random calssifier is 0.05, while for the n-grams classifier is 0.81. When $t = 0.8$, the n-grams classifier has an F1-score of ≈ 0.7, indicating that traffic analysis is a true threat to DNS privacy even in the open world scenario.

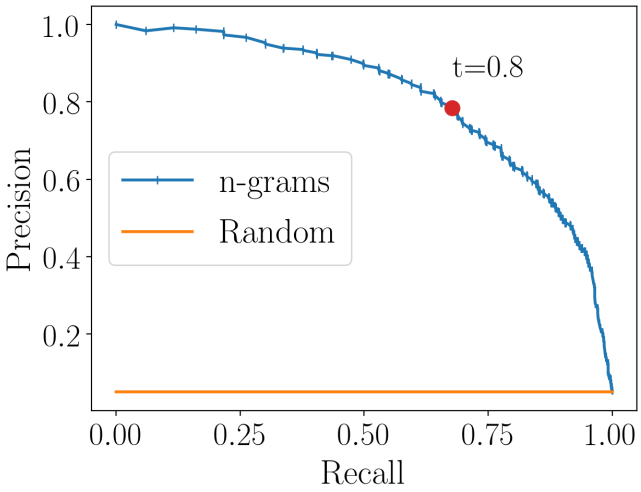


Figure 2.3: Precision-Recall ROC curve for open world classification, for the monitored class. The notches indicate the variation in threshold, t.

Comparison to web traffic fingerprinting. To understand the gain DNS fingerprinting provides to an adversary, we compare its effectiveness to that of web traffic fingerprinting. We also evaluate the suitability of n-grams and traditional website fingerprinting features

to both fingerprinting problems. We compare it to state-of-the-art attacks that use different features: the k-Fingerprinting attack, by Hayes and Danezis [21], that considers a comprehensive set of features used in the website fingerprinting literature; CUMUL, by Panchenko *et al.* [19] which focuses on packets' lengths and order through cumulative features; and Deep Fingerprinting (DF), an attack based on deep convolutional neural networks [22]. In this comparison, we consider a closed-world of 700 websites (WEB dataset) and use a random forest with the same parameters as classification algorithm. We evaluate the performance of the classifiers on only DoH traffic (DoH-only), only HTTPS traffic corresponding to web content traffic (Web-only), and a mix of the two in order to verify the claim in the RFC that DoH [9] holds great potential to thwart traffic analysis. Table 2.3 shows the results.

First, we find that for DNS traffic, due to its chatty characteristics, n-grams provide more than 10% performance improvement with respect to traditional features. DF would probably achieve higher accuracy if it was trained with more data. To obtain a significant improvement, however, DF requires orders of magnitude more data. Thus, it scales worse than our attack. We also see that the most effective attacks are those made on web traffic. This is not surprising, as the variability of resources' sizes in web traffic contains more information than the small DNS packets. What is surprising is that the n-grams features *outperform* the traditional features *also* for website traffic. Finally, as predicted by the standard, if DoH and HTTPS are sent on the same TLS tunnel and cannot be separated, both set of features see a decrease in performance. Still, n-grams outperforms traditional features, with a ∼10% improvement.

In summary, the best choice for an adversary with access to the isolated HTTPS flow is to analyse that trace with our novel n-grams features. However, if the adversary is in 'a different path than the communication between the initiator and the recipient' [6] where she has access to DNS, or is limited in resources (see below), the DNS encrypted flow provides comparable results.

Table 2.3: F1-Score of the n-grams, k-Fingerprinting, CUMUL and DF features for different subsets of traffic: only DoH traffic (DoH-only), only HTTPS traffic corresponding to web traffic (Web-only) and mixed (DoH+Web).

	DoH-only	Web-only	DoH + Web
n-grams	0.87	0.99	0.88
k-Fingerprinting [21]	0.74	0.95	0.79
CUMUL [19]	0.75	0.92	0.77
DF [22][6]	0.51	0.94	0.75

[6]We evaluate DF's accuracy following the original paper, i.e., using validation and test sets, instead of 10-fold cross-validation.

Adversary's effort. An important aspect to judge the severity of traffic analysis attacks is the effort needed regarding data collection to train the classifier [31]. We study this effort from two perspectives: amount of samples required – which relates to the time needed to prepare the attack, and volume of data – which relates to the storage and processing requirements for the adversary.

We first look at how many samples are required to train a well-performing classifier. We see in Table 2.4 that there is a small increase between 10 and 20 samples, and that after 20 samples, there are diminishing returns in increasing the number of samples per domain. This indicates that, in terms of number of samples, the collection effort to perform website identification on DNS traces is *much smaller* than that of previous work on web traffic analysis: Website fingerprinting studies in Tor report more than 10% increase between 10 and 20 samples [33] and between 2% and 10% between 100 and 200 samples [55, 22].

We believe the reason why fingerprinting DoH requires fewer samples per domain is DoH's lower intra-class variance with respect to encrypted web traffic. This is because sources of large variance in web traffic, such as the presence of advertisements which change accross visits thereby varying the sizes of the resources, does not show in DNS traffic for which the same ad-network domains are resolved [72].

Table 2.4: Classifier performance for different number of samples in the LOC1 dataset averaged over 10-fold cross validation (standard deviations less than 1%).

Number of samples	Precision	Recall	F1-score
10	0.873	0.866	0.887
20	0.897	0.904	0.901
40	0.908	0.914	0.909
100	0.912	0.916	0.913

In terms of volume of data required to launch the attacks, targeting DoH flows also brings great advantage. In the WEB dataset, we observe that web traces have a length of 1.842 MB ± 2.620 MB, while their corresponding DoH counterpart only require 0.012 MB ± 0.012 MB. While this may not seem a significant difference, when we look at the whole dataset instead of individual traces, the HTTPS traces require 73GB while the DoH-only dataset fits in less than 1GB (0.6GB). This is because DNS responses are mostly small, while web traffic request and responses might be very large and diverse (*e.g.*, different type of resources, or different encodings).

In our experiments, to balance data collection effort and performance, we collected 60 samples per domain for all our datasets. For the unmonitored websites in the open world we collected just three samples per domain (recall that we do not identify unmonitored websites).

2.5.3 DNS Fingerprinting Robustness

In practice, the capability of the adversary to distinguish websites is very dependent on differences between the setup for training data collection and the environmental conditions at attack time [73]. We present experiments exploring three environmental dimensions: time, space, and infrastructure.

Robustness over time

DNS traces vary due to the dynamism of webpage content and variations in DNS responses (e.g., service IP changes because of load-balancing). To understand the impact of this variation on the classifier, we collect data LOC1 for 10 weeks from the end of September to the beginning of November 2018. We divide this period into five intervals, each containing two consecutive weeks, and report in Table 2.5 the F1-score of the classifier when we train the classifier on data from a single interval and use the other intervals as test data (0 weeks old denotes data collected in November). In most cases, the F1-score does not significantly decrease within a period of 4 weeks. Longer periods result in significant drops – more than 10% drop in F1-score when the training and testing are separated by 8 weeks.

Table 2.5: F1-score when training on the interval indicated by the row and testing on the interval in the column (standard deviations less than 1%). We use 20 samples per webpage (the maximum number of samples collected in all intervals).

F1-score	0 weeks old	2 weeks old	4 weeks old	6 weeks old	8 weeks old
0 weeks old	0.880	0.827	0.816	0.795	0.745
2 weeks old	0.886	0.921	0.903	0.869	0.805
4 weeks old	0.868	0.898	0.910	0.882	0.817
6 weeks old	0.775	0.796	0.815	0.876	0.844
8 weeks old	0.770	0.784	0.801	0.893	0.906

This indicates that, to obtain best performance, an adversary with the resources of a university research group would need to collect data at least once a month. However, it is unlikely that DNS traces change drastically. To account for gradual changes, the adversary can perform continuous collection and mix data across weeks. In our dataset, if we combine two- and three-week-old samples for training; we observe a very small decrease in performance. Thus, a continuous collection strategy can suffice to maintain the adversary's performance without requiring periodic heavy collection efforts.

Robustness across locations

DNS traces may vary across locations due to several reasons. First, DNS lookups vary when websites adapt their content to specific geographic regions. Second, popular resources cached by resolvers vary across regions. Finally, resolvers and CDNs use geo-location methods to load-balance requests, *e.g.*, using anycast and EDNS [74, 75].

We collect data in three locations, two countries in Europe (LOC1 and LOC2) and a third country in Asia (LOC3). Table 2.6 (leftmost) shows the classifier performance when crossing these datasets for training and testing. When trained and tested on the same location unsurprisingly the classifier yields results similar to the ones obtained in the base experiment. When we train and test on different locations, the F1-score decreases between a 16% and a 27%, the greatest drop happening for the farthest location, LOC3, in Asia.

Interestingly, even though LOC2 yields similar F1-Scores when cross-classified with LOC1 and LOC3, the similarity does not hold when looking at Precision and Recall individually. For example, training on LOC2 and testing on LOC1 results in around 77% Precision and Recall, but training on LOC1 and testing on LOC2 gives 84% Precision and 65% Recall. Aiming at understanding the reasons behind this asymmetry, we build a classifier trained to separate websites that obtain high recall (top 25% quartile) and low recall (bottom 25% quartile) when training with LOC1 and LOC3 and testing in LOC2. A feature importance analysis on this classifier showed that LOC2's low-recall top features have a significantly lower importance in LOC1 and LOC2. Furthermore, we observe that the intersection between LOC1 and LOC3's relevant feature sets is slightly larger than their respective intersections with LOC2. While it is clear that the asymmetry is caused by the configuration of the network in LOC2, its exact cause remains an open question.

Robustness across infrastructure

In this section, we study how the DoH resolver and client, and the user platform affect influence the attack's performance.

Influence of DoH Resolver. We study two commercial DoH resolvers, Cloudflare's and Google's. Contrary to Cloudflare, Google does not provide a stand-alone DoH client. To keep the comparison fair, we instrument a new collection setting using Firefox in its *trusted recursive resolver* configuration with both DoH resolvers.

Table 2.6 (center-left) shows the result of the comparison. As expected, training and testing on the same resolver yields the best results. As in the location setting, we observe an asymmetric decrease in one of the directions: training on GOOGLE dataset and

Table 2.6: Performance variation changes in location and infrastructure (F1-score, standard deviations less than 2%).

Location	LOC1	LOC2	LOC3
LOC1	0.906	0.712	0.663
LOC2	0.748	0.908	0.646
LOC3	0.680	0.626	0.917

Client	CLOUD	CL-FF	LOC2
CLOUD	0.885	0.349	0.000
CL-FF	0.109	0.892	0.069
LOC2	0.001	0.062	0.908

Resolver	GOOGLE	CLOUD
GOOGLE	0.880	0.129
CLOUD	0.862	0.885

Platform	DESKTOP	RPI
DESKTOP	0.8802	0.0003
RPI	0.0002	0.8940

attacking CLOUD results in 13% F1-score, while attacking GOOGLE with a classifier trained on CLOUD yields similar results as training on GOOGLE itself.

To investigate this asymmetry we rank the features according their importance for the classifiers. For simplicity, we only report the result on length unigrams, but we verified that our conclusions hold when considering all features together. Figure 2.4 shows the top-15 most important features for a classifier trained on Google's resolver (left) and Cloudflare's (right). The rightmost diagram of each column shows the importance of these features on the other classifier. Red tones indicate high importance, and dark colors represent irrelevant features. Grey indicates that the feature is not present.

We see that the most important features in Google are either not important or missing in Cloudflare (the right column in left-side heatmap is almost gray). As the missing features are very important, they induce erroneous splits early in the trees, and for a larger fraction of the data, causing the performance drop. However, only one top feature in the classifier trained on Cloudflare is missing in Google, and the others are also important (right column in right-side heatmap). Google does miss important features in Cloudflare, but they are of little importance and their effect on performance is negligible.

Influence of end user's platform. We collect traces for the 700 top Alexa webpages on a Raspberry Pi (RPI dataset) and an Ubuntu desktop (DESKTOP dataset), both from LOC1. We see in Table 2.6 (center-right) that, as expected, the classifier has good performance when the training and testing data come from the same platform. However, it drops to almost zero when crossing the datasets.

When taking a closer look at the TLS record sizes from both platforms we found that TLS records in the DESKTOP dataset are on average 7.8 bytes longer than those in RPI (Figure 2.5). We repeated the cross classification after adding 8 bytes to all RPI TLS

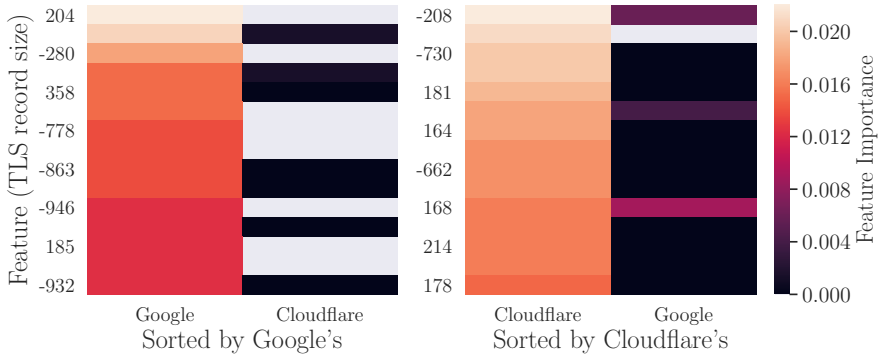


Figure 2.4: Top 15 most important features in Google's and Cloudflare's datasets. On the left, features are sorted by the results on Google's dataset and, on the right, by Cloudflare's.

record sizes (Table 2.7). Even though the classifiers do not reach the base experiment's performance, we see a significant improvement in cross-classification F1-score to 0.614 when training on DESKTOP and testing on RPI, and 0.535 when training on RPI and testing on DESKTOP.

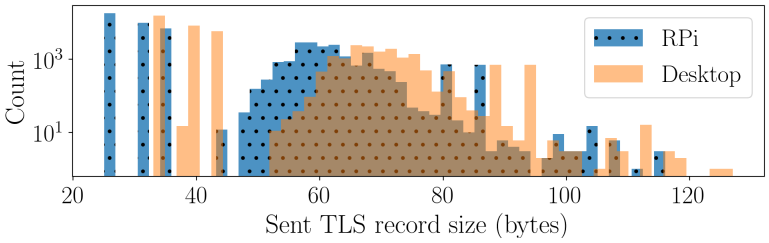


Figure 2.5: Distribution of user's sent TLS record sizes in platform experiment.

Table 2.7: Improvement in cross platform performance when removing the shift (standard deviation less than 1%).

Train	Test	Precision	Recall	F-score
DESKTOP	RPI	0.630	0.654	0.614
RPI	DESKTOP	0.552	0.574	0.535

Influence of DNS client. Finally, we consider different client setups: Firefox's trusted recursive resolver or TRR (CLOUD), Cloudflare's DoH client with Firefox (CL-FF) and Cloudflare's DoH client with Chrome (LOC2). We collected these datasets in location LOC2 using Cloudflare's resolver.

Table 2.6 (rightmost) shows that the classifier performs as expected when trained and tested on the same client setup. When the setup changes, the performance of the classifier

drops dramatically, reaching zero when we use different browsers. We hypothesize that the decrease between CL-FF and LOC2 is due to differences in the implementation of the Firefox's built-in and Cloudflare's standalone DoH clients.

Regarding the difference when changing browser, we found that Firefox' traces are on average 4 times longer than Chrome's. We looked into the unencrypted traffic to understand this difference. We used the man-in-the-middle proxy and the Lekensteyn's scripts to decrypt DoH captures for Firefox configured to use Cloudflare's resolver. For Google's resolver, we man-in-the-middle a curl-doh client[7], which also has traces substantially shorter than Firefox. We find that Firefox, besides resolving domains related to the URL we visit, also issues resolutions related to OSCP servers, captive portal detection, user's profile/account, web extensions, and other Mozilla servers. As a consequence, traces in CL-FF and CLOUD datasets are substantially larger and contain different TLS record sizes than any of our other datasets. We conjecture that Chrome performs similar requests, but since traces are shorter we believe the amount of checks seems to be smaller than Firefox's.

Robustness Analysis Takeaways

Our robustness study shows that to obtain best results across different configurations the adversary needs i) to train a classifier for each targeted setting, and ii) to be able to identify her victim's configuration. Even if the adversary needs to train a classifier for each setting, this is less costly in the case of DoH fingerprinting as compared to website fingerprinting due to the training requiring significantly less data. Kotzias *et al.* demonstrated that identifying client or resolver is possible, for instance examining the IP (if the IP is dedicated to the resolver), or fields in the ClientHello of the TLS connection (such as the Server Name Indication (SNI), cipher suites ordering, etc.) [76]. Even if TLS 1.3 encrypts some of these headers and are thus not directly available to the adversary, we found that characteristics of traffic itself are enough to identify a resolver. We built classifiers to distinguish resolver and client based on the TLS record length. We can identify resolvers with 95% accuracy, and we get no errors (100% accuracy) when identifying the client.

When analyzing traces, we observed customization by the DNS providers that are not part of the standard. For example, Cloudflare includes HTTP headers such as CF-RAY to trace a request through its network. Such customization can lead to differences in the traffic among different providers and have an impact on the classification.

Regarding users' platform, we see little difference between desktops, laptops, and servers

[7] https://github.com/curl/doh

in Amazon Web Services. However, we observe a difference between these and constrained devices, such a Raspberry Pi. This different results on a drop in accuracy when training a classifier on one type and deploying it on the other.

Finally, our longitudinal analysis reveals that, keeping up with the changes in DNS traces can be done at low cost by continuously collecting samples and incorporating them to the training set.

Survivors and Easy Preys. We study whether there are websites that are particularly good or bad at evading fingerprinting under any circumstance. We compute the mean F1-Score across all configurations as an aggregate measure of the attack's overall performance. We plot the CDF of the distribution of mean F1-scores over the websites in Figure 2.6. This distribution is heavily skewed: there are up to 15% of websites that had an F1-Score equal or lower than 0.5 and more than 50% of the websites have a mean F1-Score equal or lower than 0.7.

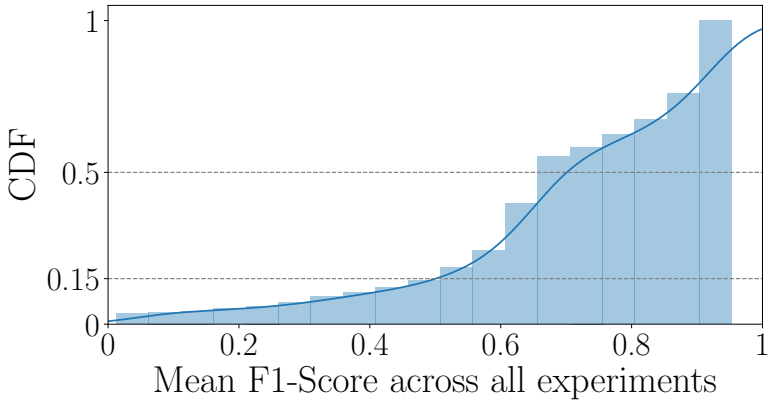


Figure 2.6: Cumulative Distribution Function (CDF) of the per-class mean F1-Score.

We rank sites by lowest mean F1-Score and lowest standard deviation. At the top of this ranking, there are sites that *survived* the attack in all configurations. Among these survivors, we found Google, as well as sites giving errors, that get consistently misclassified as each other. We could not find any pattern in the website structure or the resource loads that explains why other sites with low F1 survive. At the bottom of the ranking, we find sites with long domain names, with few resources and that upon visual inspection present low variability.

Data Pollution Our experiments evaluate the scenario where a user visits a single webpage at a time. However, in many cases the user might visit more than one page at a time, might have multiple tabs open, or might have applications running in the background. These scenarios could *pollute* the traces by mixing the DoH traffic from

multiple webpages. Since it is non-trivial for an adversary to split traffic belonging to different webpages, such pollution can have an impact on the attack effectiveness. However, at the time of writing there is no standard way in the literature to study the impact of polluted DNS traffic. Designing a method to evaluate this effect is beyond the scope of this work.

Pollution could also be triggered on the resolver side, if the resolver serves both DoH and non-DoH traffic on the same connection. The resolvers studied in our experiments, Cloudflare and Google, only host one webpage: `one.one.one.one` and `dns.google.com`, respectively. By analyzing the traffic of a user visiting these webpages, we observe that the TLS flows carrying the web content and the DoH flows corresponding to the domain resolution can be trivially distinguished using traffic features such as the number of packets (HTTPS traces are longer than DoH traces) and the packet sizes (DNS responses are much smaller than the resources sent over HTTPS). We do not expect this kind of pollution to be an obstacle for the adversary.

2.6 DNS Defenses against fingerprinting

In this section, we evaluate existing techniques to protect encrypted DNS against traffic analysis. Table 2.8 summarizes the results. We consider the following defenses:

EDNS(0) Padding. EDNS (Extension mechanisms for DNS) is a specification to increase the functionality of the DNS protocol [77]. It specifies how to add *padding* [16], both on DNS clients and resolvers, to prevent size-correlation attacks on encrypted DNS. The recommended padding policy is for clients to pad DNS requests to the nearest multiple of 128 bytes, and for resolvers to pad DNS responses to the nearest multiple of 468 bytes [78].

Cloudflare's DoH client provides functionality to set EDNS(0) padding to DNS queries, leaving the specifics of the padding policy to the user. We modify the client source code to follow the padding strategy above. Google's specification also mentions EDNS padding. However, we could not find any option to activate this feature, thus we cannot analyze it.

In addition to the EDNS(0) padding, we wanted to see whether simple user-side measures that alter the pattern of requests, such as the use of an ad-blocker, could be effective countermeasures. We conduct an experiment where we use Selenium with a Chrome instance with the AdBlock Plus extension installed. We do this with DoH requests and responses padded to multiples of 128 bytes.

Upon responsible disclosure of this work, Cloudflare's added padding to the responses of their DoH resolver. However, when analyzing the collected data we discover that they do *not* follow the recommended policy. Instead of padding to multiples of 468 bytes, Cloudflare's resolver pads responses to multiples of 128 bytes, as recommended for DoH clients. In order to also evaluate the recommended policy, we set up an HTTPS proxy (*mitmproxy*) between the DoH client and the Cloudflare resolver. The proxy intercepts responses from Cloudflare's DoH resolver, strips the existing padding, and pads responses to the nearest multiple of 468 bytes.

As we show below, none of these padding strategies completely stops traffic analysis. To understand the limits of protection of padding, we simulate a setting in which padding is perfect, *i.e.*, *all* records have the same length and the classifier cannot exploit the TLS record size information. To simulate this setting, we artificially set the length of all packets in the dataset to 825, the maximum size observed in the dataset.

DNS over Tor. Tor is an anonymous communication network. To protect the privacy of its users, Cloudflare set up a DNS resolver that can be accessed using Tor. This enables users to not reveal their IP to the resolver when doing look-ups. To protect users' privacy, Tor re-routes packets through so-called onion routers to avoid communication tracing based on IP addresses; and it packages content into constant-size cells to prevent size-based analysis. These countermeasures have so far not been effective to protect web traffic [19, 20, 21, 22]. We study whether they can protect DNS traffic.

Table 2.8: Classification results for countermeasures.

Method	**Precision**	**Recall**	**F1-score**
EDNS0-128	0.710 ± 0.005	0.700 ± 0.004	0.691 ± 0.004
EDNS0-128-adblock	0.341 ± 0.013	0.352 ± 0.011	0.325 ± 0.011
EDNS0-468	0.452 ± 0.007	0.448 ± 0.006	0.430 ± 0.007
Perfect Padding	0.070 ± 0.003	0.080 ± 0.002	0.066 ± 0.002
DNS over Tor	0.035 ± 0.004	0.037 ± 0.003	0.033 ± 0.003
DNS over TLS	0.419 ± 0.008	0.421 ± 0.007	0.395 ± 0.007

Results. Our first observation is that EDNS0 padding is not as effective as expected. Adding more padding, as recommended in the specification, does provide better protection, but still yields an F1-score of 0.45, six orders of magnitude greater than random guessing. Interestingly, usage of an ad-blocker helps as much as increasing the padding, as shown by the EDNS0-128-adblock experiment. As shown below, Perfect Padding would actually deter the attack, but at a high communication cost.

As opposed to web traffic, where website fingerprinting obtains remarkable performance [63,

21, 22], Tor is very effective in hiding the websites originating a DNS trace. The reason is that DNS look-ups and responses are fairly small. They fit in one, at most two, Tor cells which in turn materialize in few observed TLS record sizes. As a result, it is hard to find features unique to a page. Also, DNS traces are shorter than normal web traffic, and present less variance. Thus, length-related features, which have been proven to be very important in website fingerprinting, only provide a weak 1% performance improvement.

Even though Perfect Padding and DNS over Tor offer similar performance, when we look closely at the misclassified webpages, we see that their behavior is quite different. For Tor, we observe misclassifications cluster around six different groups, and in Perfect Padding they cluster around 12 (different) groups (see the Appendix for confusion graphs). For both cases, we tested that it is possible to build a classifier that identifies the cluster a website belongs to with reasonable accuracy. This means that despite the large gain in protection with respect to EDNS(0), the effective anonymity set for a webpage is much smaller than the total number of webpages in the dataset.

Finally, we evaluate defenses' communication overhead. For each countermeasure, we collect 10 samples of 50 webpages, with and without countermeasures, and measure the difference in total volume of data exchanged between client and resolver. We see in Figure 2.7 that, as expected, EDNS0 padding (both 128 and 468) incur the least overhead, but they also offer the least protection. DNS over Tor, in turn, offers lower overhead than Perfect Padding.

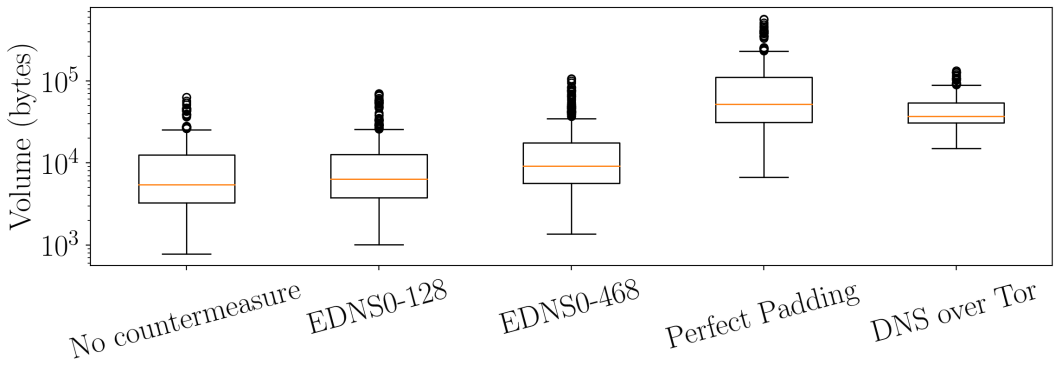


Figure 2.7: Total traffic volume with and without defenses.

Comparison with DNS over TLS (DoT). Finally, we compare the protection provided by DNS over HTTPS and over TLS, using the DOT dataset. As in DoH, Cloudflare's DoT resolver implements EDNS0 padding of DNS responses to a multiple of 128 bytes. The `cloudflared` DoT client, however, does not support DoT traffic. Thus, we use the Stubby client to query Cloudflare's DoT padded to a multiple of 128 bytes.

Our results (shown in the last row of Table 2.8) indicate that DoT offers much better protection than DoH – about 0.3 reduction in F1-score. We plot a histogram of sizes of the sent and received TLS records for both DoH and DoT for 100 webpages (Figure 2.8). We observe that DoT traffic presents much less variability in TLS record sizes than DoH.

To gain insights into the origin of this variability, we collected and decrypted DoT and DoH traffic for visits to a handful of websites. We used a modified Stubby client[8] to decrypt the DoT traffic, and used the tools described in Section 2.5.1 to decrypt the DoH traces. For each website, we used Firefox and Stubby (DoT setting) and, immediately after finishing loading the page, we restarted Firefox with a clean profile and visited using Firefox's native DoH client (DoH setting).

Upon inspection of the traces, we observe that DoH traces contain requests for A and AAAA records while DoT traces only have requests for A records. This shows how different implementations of the client can generate significantly different traffic, potentially affecting the performance of the attack. This also explains why we observe a greater number of TLS record sizes in Figure 2.8. We also find differences in the domains that are resolved. However, most of the differences are domains that occur independently of the protocol and, thus, cannot explain the difference in attack performance between the two protocols. The most relevant difference is that, excluding AAAA queries and responses, DoT traces have fewer average number of DNS messages than DoH traces for the same website. In addition, the TLS records in DoT traces are larger than DoH's[9]. We acknowledge that although we identify possible causes for the variability of DoH's TLS record sizes, our observations are not conclusive on whether this variability accounts for the better performance of the attack on DoT than DoH.

2.7 Censorship on encrypted DNS

DNS-based blocking is a wide-spread method of censoring access to web content. Censors inspect DNS look-ups and, when they detect a blacklisted domain, they either reset the connection or inject their own DNS response [79]. The use of encryption in DoH precludes content-based lookup identification. The only option left to the censor is to block the resolver's IP. While this would be very effective, it is a very aggressive strategy, as it would prevent users from browsing any web. Furthermore, some DoH resolvers, such as Google's, do not necessarily have a dedicated IP. Thus, blocking their IP may also affect

[8] https://github.com/saradickinson/getdns/tree/1.5.2_add_keylogging

[9] This might seem counter-intuitive as DoH has HTTP headers that DoT does not have. However, DoH's RFC allows the use of HTTP compression (and we observe its use in our dataset), while DoT's RFC recommends against the use of TLS compression.

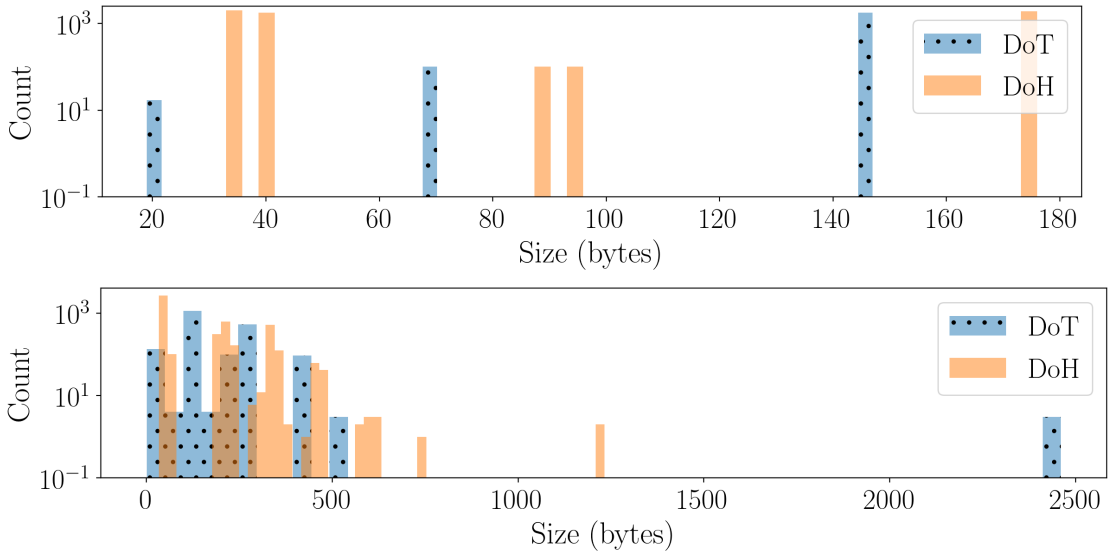


Figure 2.8: Histogram of sent (top) and received (bottom) TLS record sizes for DoT and DoH.

other services.

An alternative for the censor is to use traffic analysis to identify the domain being looked up. In this section, we study whether such an approach is feasible. We note that to block access to a site, a censor not only needs to identify the lookup from the encrypted traffic, but also needs to do this as soon as possible to prevent the user from downloading any content.

2.7.1 Uniqueness of DoH traces

In order for the censor to be able to uniquely identify domains given DoH traffic, the DoH traces need to be unique. In particular, to enable early blocking, *the first packets* of the trace need to be unique.

To study the uniqueness of DoH traffic, let us model the set of webpages in the world as a random variable W with sample space Ω_W; and the set of possible network traces generated by those websites as a random variable S with sample space Ω_S. A website's trace w is a sequence of non-zero integers: $(s_i)_{i=1}^{n}, s_i \in \mathbb{Z} \setminus \{0\}, n \in \mathbb{N}$, where s_i represents the size (in bytes) of the i-th TLS record in the traffic trace. Recall that its sign represents the direction – negative for incoming (DNS to client) and positive otherwise. We denote partial traces, *i.e.*, only the l first TLS records, as S_l.

We measure *uniqueness* of partial traces using the conditional entropy $H(W \mid S_l)$, defined as:

$$H(W \mid S_l) = \sum_{\forall o \in \Omega_{S_l}} \Pr[S_l = o] H(W \mid S_l = o)\,.$$

Here, $H(W \mid S_l = o)$ is the Shannon entropy of the probability distribution $\Pr[W \mid S_l = o]$. This probability describes the likelihood that the adversary guesses websites in W given the observation o. The conditional entropy $H(W \mid S_l)$ measures how distinguishable websites in Ω_W are when the adversary has only observed l TLS records. When this entropy is zero, sites are perfectly distinct. For instance, if the first packet of every DoH trace had a different size, then the entropy $H(W \mid S_1)$ would be 0.

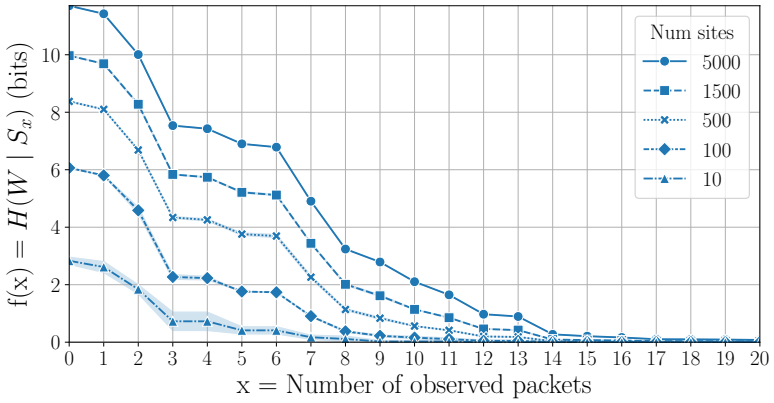


Figure 2.9: Conditional entropy $H(W \mid S_l)$ given partial observations of DoH traces for different world sizes ($|\Omega_W| = \{10, 100, 500, 1500, 5000\}$). Each data point is averaged over 3 samples.

We compute the conditional entropy for different world sizes $|\Omega_W|$ and partial lengths l, using traces from the OW dataset. We show the result in Figure 2.9. Every point is an average over 3 samples of $|\Omega_W|$ webs. These webs are selected uniformly at random from the dataset. The shades represent the standard deviation across the 3 samples.

Unsurprisingly, as the adversary observes more packets, the traces become more distinguishable and the entropy decreases. For all cases, we observe a drop of up to 4 bits within the first four packets, and a drop below 0.1 bits after 20 packets. As the world size increases, the likelihood of having two or more websites with identical traces increases, and thus we observe a slower decay in entropy. We also observe that, as the world size increases the standard deviation becomes negligible. This is because the OW dataset contains 5,000 websites. Thus, as the size of the world increases, the samples of Ω_W contain more common websites.

Even when considering 5,000 pages, the conditional entropy drops below 1 bit after 15 packets. This means that after 15 packets have been observed, there is one domain whose probability of having generated the trace is larger than 0.5. In our dataset 15 packets

is, on average, just 15% of the whole trace. Therefore, on average, the adversary only needs to observe the initial 15% of a DoH connection to determine a domain with more confidence than taking a random guess between two domains. We observe that as the number of pages grows, the curves are closer to each other indicating that convergence on uniqueness after the 15-th packet is likely to hold in larger datasets.

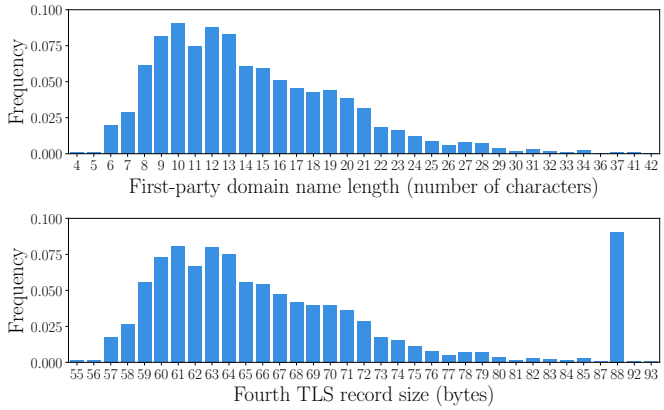


Figure 2.10: Histograms for domain name length (top) and fourth TLS record length (bottom) in the LOC1 dataset, for 10 samples (normalized over the total sum of counts).

The importance of the fourth packet. We hypothesized that the large drop by the fourth packet might be caused by one of these records containing the DNS lookup. As these traces do not contain padding, the domain length would be directly observable in the trace. We verify this hypothesis by comparing the frequency of appearance of the domain's and outgoing fourth record's length (incoming records cannot contain a lookup). We discard TLS record sizes corresponding to HTTP2 control messages, e.g., the size "33" which corresponds to HTTP2 acknowledgements and outliers that occurred 5% or less times. However, We kept size "88" even though it appears too often, as it could be caused by queries containing 37-characters-long domain names.

We represent the frequency distributions in Figure 2.10. We see that the histogram of the sizes of the fourth TLS record in our dataset is almost identical to the histogram of domain name lengths, being the constant difference of 51 bytes between the two histograms the size of the HTTPS header. This confirms that the fourth packet often contains the first-party DoH query. In some traces the DoH query is sent earlier, explaining the entropy decrease starting after the second packet. On decrypting the connection, we observe that packets prior to the the query contain HTTP/2 control information.

2.7.2 Censor DNS-blocking strategy

We study the collateral damage, *i.e.*, how many sites are affected when a censor blocks one site after a traffic-analysis based inference. We assume that upon decision, the censor uses standard techniques to block the connection [80].

High-confidence blocking. To minimize the likelihood of collateral damage, the adversary could wait to see enough packets for the conditional entropy to be lower than one bit. This requires waiting for, on average, 15% of the TLS records in the DoH connection before blocking. As those packets include the resolution to the first-party domain, the client can download the content served from this domain. Yet, the censor can still disrupt access to subsequent queried domains (subdomains and third-parties). This is a strategy already used in the wild as a stealthy form of censorship [80].

To avoid this strategy, the clients could create one connection per domain lookup, thereby mixing connections belonging to the censored page and others. At the cost of generating more traffic for users and resolvers, this would force the censor to drop all DoH connections originating from a user's IP or throttle their DoH traffic, causing more collateral damage.

Block on first DoH query. A more aggressive strategy is to drop the DoH connection before the first DoH response arrives. While this guarantees that the client cannot access any content, not even `index.html`, it also results in all domains with same name length being censored.

In order to understand the collateral damage incurred by domain length based blocking relying on the fourth packet, we compare the distribution of domain name lengths in the Alexa top 1M ranking (see Fig. 2.11) with the distribution of domain names likely to be censored in different countries. For the former we take the global ranking as per-country rankings only provide 500 top websites which are not enough for the purpose of our experiments and, for some countries, the lists are not even available. We take the test lists provided by Citizen Labs [81]. These lists contain domains that regional experts identify as likely to be censored. While appearance in these lists does not guarantee that the domains are actually censored, we believe that they capture potential censor behavior.

We take five censorship-prone countries as examples: Turkmenistan, Iran, Saudi Arabia, Vietnam, and China. Our analysis of collateral damage is relative among the countries considered in our study. Since the Alexa list contains only domains, we extract the domains from the URLs appearing in Citizen Labs' test lists. Table 2.9 shows the total number of domains in each list and the number of those domains that are not present in

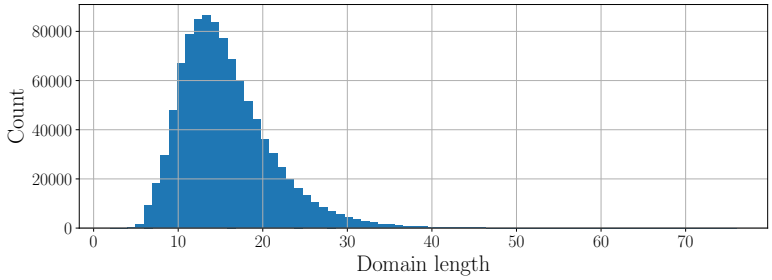


Figure 2.11: Distribution of domain length in the Alexa top 1M.

the Alexa-1M list. We observe that at most 51% (China) of the domains appear in the ranking. For the other countries the ratio is between 21% and 32%. This indicates that the potentially censored domains themselves are mostly not popular.

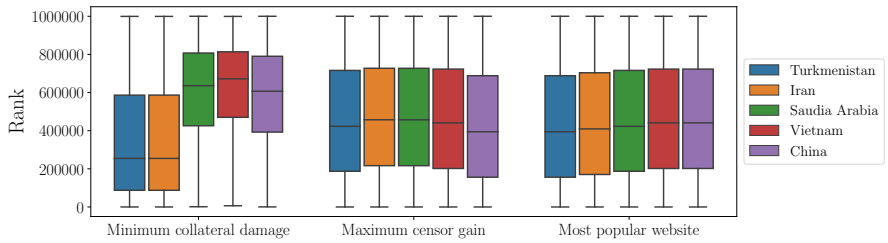


Figure 2.12: Collateral damage in terms of ranking for three blocking strategies: minimum collateral damage set size, maximum censor gain, most popular website.

Even if the censor blocks unpopular domains, there are two side effects of blocking based on domain lengths. First, there will be some collateral damage: allowed websites with the same domain length will also be blocked. Second, there will be a gain for the censor: other censored websites with the same domain length are blocked at the same time.

We study the trade-off between collateral damage and censor gain. The minimum collateral damage is attained when the censor blocks domains of length 6 (such as `ft.com`), resulting on 1,318 affected websites. The maximum damage happens when censoring domains of size 10 (such as `google.com`) which affects other 66,923 websites, domains of size 11 (such as `youtube.com`) which affects other 78,603 websites, domains of size 12 (such as `facebook.com`) which affects other 84,471 websites, domains of size 13 (such as `wikipedia.org`) which affects other 86,597 websites, and domains of size 14 (such as `torproject.org`, which affects other 83,493 websites. This means that, in the worst case, collateral damage is at most 8.6% of the top 1M list.

To understand the censor gain, we consider Iran as an example. The maximum gain is 97, for domain length of 13, *i.e.*, when the censor blocks domains of length 13, it blocks

97 domains in the list. At the same time, it results in large collateral damage (86,557 websites). The minimum gain is obtained for domain length 5, which only blocks one website, but causes small collateral damage (1,317 domains). If Iran blocks the most popular domain in the list (google.com), this results in a collateral damage of 66,887 websites.

The volume of affected websites is of course representative of damage, but it is not the only factor to take into account. Blocking many non-popular domains that are in the bottom of the Alexa list is not the same as blocking a few in the top-100. We show in Figure 2.12 box plots representing the distribution of Alexa ranks for three blocking strategies: minimum collateral damage, maximum censor gain, and blocking the most popular web. For each country, these strategies require blocking different domain lengths. Thus, in terms of volume, the damage is different but in the order of the volumes reported above. We observe that the minimum collateral damage strategy results in different impact depending on the country. Although for all of them the volume is much lower than for the other strategies, in Turkmenistan and Iran the median rank of the blocked websites is much lower than those in the other strategies, indicating that this strategy may have potentially higher impact than blocking more popular or high-gain domains. On the positive side, minimum collateral damage in Saudi Arabia, Vietnam, and China, mostly affects high-ranking websites. Thus, this strategy may be quite affordable in this country. The other two strategies, maximum gain and blocking the most popular website, result on a larger number of websites blocked, but their median ranking is high (above 500,000) and thus can also be considered affordable.

Table 2.9: Number of domains in each censorship test list and their presence in the Alexa top 1M.

	Turkmenistan	**Iran**	**S. Arabia**	**Vietnam**	**China**
Censored domains	344	877	284	274	191
Not in Alexa-1M	246	600	219	218	94

In conclusion, our study shows that while block on first DoH query is an aggressive strategy, it can be affordable in terms of collateral damage. More research is needed to fine-tune our analysis, e.g., with access to large per-country rankings, or exact lists of blacklisted domains.

2.8 Looking ahead

We have shown that, although it is a great step for privacy, encrypting DNS does not completely prevent monitoring or censorship. Current padding strategies have great potential to prevent censorship, but our analysis shows that they fall short when it comes

to stopping resourceful adversaries from monitoring users' activity on the web.

Our countermeasures analysis hints that the path towards full protection is to eliminate size information. In fact, the repacketization in constant-size cells offered by Tor provides the best practical protection. Tor, however, induces a large overhead both in the bandwidth required to support onion encryption, as well as in download time due to the rerouting of packets through the Tor network.

We believe that the most promising approach to protect DoH is to have clients mimicking the repacketization strategies of Tor, without replicating the encryption scheme or re-routing. This has the potential to improve the trade-off between overhead and traffic analysis resistance. A complementary strategy to ease the achievement of constant-size flows, is to rethink the format of DNS queries and its headers. Reducing the number of bits required for the headers would make it easier to fit queries and responses in one constant-size packet with small overhead.

Besides protection against third party observers, it is important that the community also considers protection from the resolvers. The current deployment of DoH, both at the resolvers and browsers, concentrates all look-ups among a small number of actors that can observe the behavior of users. More research in the direction of Oblivious DNS [82] is needed to ensure that no parties can become main surveillance actors.

3 Help needed! Understanding QUIC PADDING-based defenses against website fingerprinting

This chapter is based on the article:

"Help Needed! Understanding QUIC PADDING-based defenses against website fingerprinting.", S. Siby, L. Barman, C. A. Wood, M. Fayed, N.Sullivan, C. Troncoso, *to be submitted*, 2022.

3.1 Introduction

New standardization efforts have greatly increased the privacy of web traffic: *e.g.*, TLS Encrypted Client Hello (ECH) [83] to encrypt Server Name Indication (SNI), or (Oblivious) DNS-over-HTTPS [9, 84] and DNS-over-TLS [8] to encrypt DNS queries. Yet, encryption alone cannot protect users' browsing history from traffic analysis. Traffic-analysis attacks such as *website fingerprinting* (WF), enable adversaries to infer which websites a user visits from the traffic patterns (*e.g.*, volume of packets exchanged or packets' size) [85, 61, 62, 29, 86, 14].

"To provide protection against traffic analysis [...]", the working group behind QUIC, the next transport layer standard for the Web, has introduced a `PADDING` frame in the specification [15]. In this work, we assume a setting wherein websites are hosted behind content delivery networks, and use privacy-preserving protocols such as TLS ECH. Thus, the only available information to the adversary is the server IP address, which is determined by the content delivery network, and metadata such as the size of encrypted data, its timing, and its direction (sent by or to the server). In this scenario, we answer two questions:

Can we build effective traffic analysis defenses at the network layer using the `PADDING` frame? We first study whether QUIC padding configured at the transport layer as envisioned by the standard can, on its own, protect users from traffic analysis – concretely *website fingerprinting attacks*. We study adversaries with a wide range of capabilities: from powerful adversaries that can observe all communications between clients and servers, and have infinite storage and computation capability, to weaker, more realistic, scenarios in which the adversary can only observe partial traces and is restricted in its storage, computation, or bandwidth [31].

We find that:

- ✓ Network-layer defenses which build on the QUIC `PADDING` frame to hide packets' sizes or inject dummy packets are not sufficient to prevent adversaries from inferring the websites users visit. Adversaries can use global trace information (*e.g.*, the total number of packets, or the total incoming size) to recognize websites (> 92% F1-score). The adversary can successfully identify websites not just from their landing pages, but also from subpage visits, even if they have not previously encountered a visit to a specific subpage. These results hold even when the adversary uses limited information from typical network statistics (*e.g.*, NetFlow). Only when the adversary observes a very small percentage of the page does the adversary's performance decrease to close to random guessing.
- ✓ The centralization of web resources on the Internet, in particular, in the hands of Google, creates a favorable setting for the adversary. Traffic analysis on solely the timing of Google resources fetched by a web page achieves > 77% F1 score, requiring *four orders of magnitude less data than using full traces*. Moreover, the surface of attack is increased from only ASes between the client and the first-party domain host to any AS between the client and Google.

Can we inform padding algorithms so that they can effectively thwart traffic analysis? Our analysis reveals that the ineffectiveness of network-layer defenses based solely on padding QUIC traffic stems from the fact that they cannot efficiently hide the most important feature for the adversary: the total size of a given website. This is because this information is not known at the network layer. We, therefore, explore whether this information can be effectively collected at the application layer, and whether it can be used to inform padding algorithms in an effective manner:

We find that:

- ✓ Nowadays websites contain a vast amount of resources which are served, in a significant number of cases, by third parties (33% in our dataset). We demonstrate

that if all parties do not participate in defending against WF, an adversary can successfully identify pages (> 91% F1-score), leaving users vulnerable.

✓ We identify current web development practices that hinder the deployment of effective website fingerprinting defenses. We show that, unless those practices do not change, protecting users is close to impossible as it would require significant bandwidth overhead and coordination among parties under very dynamic conditions. We provide recommendations to guide future efforts and pave the way for the existence of effective defenses against website fingerprinting attacks.

Ethical considerations: We conduct traffic-analysis attacks against a deployed technology (QUIC). We do not perform any collection or analysis of real users' traffic. We only collect our own traffic, generated by an automated browser. We uncover vulnerabilities in the proposed defenses, which would put at risk, network users, if deployed. We believe that the benefits of our research, which can guide current and future standardization efforts, outweigh these risks, by avoiding deployments that could give users a false sense of security. We have performed responsible disclosure of our findings to QUIC's IETF WG.

3.2 Background & Related Work

QUIC. QUIC is a connection-oriented protocol built on top of UDP that aims to provide low-latency, multiplexed, secure communication with less head-of-line blocking and faster connection migration [10]. QUIC was standardized in May 2021 and is currently being developed by the IETF. QUIC is the transport protocol for HTTP/3. Adoption of QUIC and HTTP/3 has been rising (as of May 2022, they are used by 25% of the top 10 million websites [87]). Of particular relevance for our work is the QUIC `PADDING` Frame. The IETF QUIC draft describes it as a frame with no semantic value, that can be used to increase packets size and to provide protection against traffic analysis [15]. We investigate whether this frame is suitable to protect QUIC traffic against website fingerprinting attacks.

Website fingerprinting attacks. In website fingerprinting, an adversary aims to infer the website visited by a user by analyzing network traffic. The adversary builds a classifier trained on features obtained from website network traces. These features can be selected manually or via automatic extraction.

The most relevant attacks that rely on manual extraction are Wang *et al.*'s k-Nearest Neighbors (k-NN) classifier based on 3000 manually-selected features [54]; Panchenko *et al.*'s Support Vector Machines (SVMs)-based classifier, based on cumulative sums of

packet lengths [19]; and Hayes and Danezis [21] k-fingerprinting method (k-FP), which models web fingerprints as the leaves of a random forest built on 150 manually-selected features.

On the automatic extraction side, Rimmer *et al.* [55] use deep learning neural networks (DNNs) to produce attacks that perform as well as manual approaches. Sirinam *et al.* [22] build on Rimmer *et al.* to develop an attack that achieves high accuracy, even in the presence of defended traces. Last, Bhat *et al.* [88] propose Var-CNN, a hybrid strategy that achieves high accuracy with deep learning even in the presence of limited data. It does so by relying on ResNets trained on packet directions, packet times, and manually extracted summary statistics.

Website fingerprinting on QUIC traffic. Smith *et al.* [14] study the impact of co-existence of TCP and QUIC on the performance of website fingerprinting using k-FP and Var-CNN. They conclude that, while QUIC traffic is not difficult to fingerprint, classifiers trained on TCP traffic do not perform well on QUIC traffic, and that jointly classifying both protocols is hard. They also show that k-FP outperforms Var-CNN in the presence of QUIC. To enable comparison with the state-of-the-art we also use k-FP and Var-CNN in our evaluation. Since our aim is to understand the impact of the attack on padding defenses, we favor explainability over performance: we avoid deep-learning-based approaches [22, 89] that could yield better performance at the cost of explainability.

Website fingerprinting defenses. Dyer *et al.* [59] show that *network-layer* padding- and morphing-based countermeasures are ineffective in thwarting traffic analysis because they fail to hide coarse packet features. They propose Buffered Fixed-Length Obfuscation (BuFLO), which pads packets to a fixed size at sends them at intervals of time. BuFLO results in a huge overhead. CS-BuFLO [90] and Tamaraw [91] are more efficient, but still impractical. Works such as WTF-PAD [70] and FRONT [92] provide a better trade-off by injecting dummies at appropriate positions in a trace. WTF-PAD injects dummy packets using pre-defined distributions of inter-arrival times to detect gaps, and FRONT injects dummy packets in the front portion of traces, which are known to contain the most information for fingerprinting. Both approaches achieve low protection against deep-learning-based attacks [22, 88, 93].

Other defenses employ *adversarial perturbations* to cause deep-learning-based classifiers to misclassify traces. These defenses incur lower overhead than prior work. Mockingbird [94] applies perturbations to convert traces into a target trace. It converts traces into bursts (where a burst is a set of contiguous packets in one direction), and perturbs these bursts rather than the raw trace. To compute the perturbation, Mockingbird requires the defender to know the entire trace in advance, which is infeasible in practice. Nasr *et*

al. [95] tackle this issue with Blind, a defense that pre-computes blind perturbations that can be applied to live network traffic. Shan *et al.* [93] show that [95] offers lower protection when the adversary trains on perturbed traces. They propose Dolos, which applies adversarial patches or pre-computed sequences of dummy packets to protect network traces. Dolos utilizes a user-side secret to generate patches, making it hard for an adversary to generate the same patch, thereby reducing the risk of adversarial training.

Mockingbird, Dolos, and Blind have been developed to protect Tor traffic. Tor cells have constant size, as opposed to QUIC packets. Thus, packet sizes are not accounted for in the defenses, hindering their adoption to protect QUIC traffic. Mockingbird [94] only considers packet directionality when building dummy bursts. Thus, it is unclear how to adapt it to QUIC traffic: a burst can correspond to many QUIC packet sequences. Blind [95] does not use packet size information in their website fingerprinting evaluation. The paper contains a size perturbation technique (tailored to Tor) and uses it for their flow correlation experiments. However, even after communicating with the authors, we were unable to find where in their implementation [96] one can configure this technique, nor are there details about how to configure it to non-Tor traffic. Dolos [93] uses solely direction features to compute patches. Adapting it to QUIC would require integrating size information, for which there is no place in their implementation, with no guarantee that the patches would still be effective. Moreover, Dolos requires that a prior connection to the website has been made in order to pre-compute patches which can be used in future connections. Hence, it is predicated on information from the application layer, i.e, which website is being visited. This is in line with our findings on the importance of having the application layer informing any network layer defense (Section 3.5). Due to the challenges associated with adapting these Tor-tailored systems to our QUIC scenario, in our work, we resort to simple randomized dummies in line with WTF-PAD [70] and FRONT [92].

Finally, there are defenses at the *application layer*. Luo *et al.* [30] propose HTTP Obfuscation (HTTPOS), a client-side defense that modifies features on the TCP and HTTP layers and uses HTTP pipelining to obfuscate HTTP outgoing requests. Randomized Pipelining [97] improves this defense by randomizing the order of the HTTP requests queued in the pipeline. Subsequent works have shown HTTPOS and Randomized Pipelining to be ineffective against traffic analysis attacks [53, 54]. Cherubin *et al.* [98] developed client- and server-side defenses, LLaMA and ALPaCA respectively, tailored towards Tor onion services. These defenses only work well in scenarios with low third party content prevalence, lack of dynamic page content, and JavaScript disabled. In our work, we propose defenses inspired by ALPaCa, and study its performance in web scenarios where these assumptions do not hold (see section 3.5).

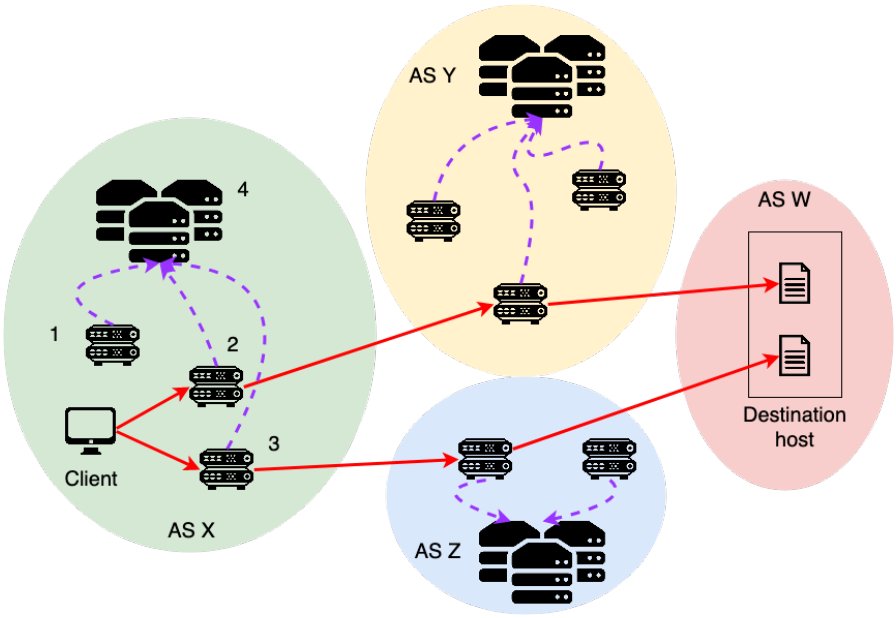


Figure 3.1: An adversary can be on any AS (X, Y, Z, or W) with vantage points on the client's traffic path (solid red arrows). The vantage points (*e.g.*, middleboxes or routers) transmit recorded data to a location that can perform traffic-analysis at scale (dotted purple arrows). If the adversary is AS X, vantage points 2 and 3 transmit recorded traffic to location 4.

3.3 Adversarial Model and Datasets

We assume a local passive eavesdropper A located at some vantage point between an honest client and an honest Web host. A observes all network traffic passing through this vantage point and records some portion of it. The goal of the adversary is to infer the domain visited by the user.

The adversary A observes IP packets. They do not possess any decryption keys, and rely only on the size and timing of the observed packets. We assume that DNS queries are done in a private manner (*e.g.*, via DoH [9] and appropriate padding [86]) and reveal no information to A. A focuses on Web traffic, and filters out packets that are not TLS or QUIC packets. Using the appropriate fields in the headers (IP addresses, ports, QUIC connection IDs), A is able to identify packets that belong to the same connection.

We call A's *observation* a collection of flows corresponding to the network connections generated when the user browses a single website. Each flow contains $[\mathrm{IP}_{\mathrm{source}}, \mathrm{IP}_{\mathrm{dest}}, p_1, p_2, \dots]$, where packets p_i are (time,size)-tuples $(t_i, \pm s_i)$. Negative sizes indicate packets from the server to the client, and positive sizes indicate packets in the opposite direction.

Vantage Points. Following prior work [99, 100, 101] we consider each AS on the path of

the client traffic as a realistic adversary. Each AS' middlebox, router, or switch that routes traffic from a client is a potential vantage point for the adversary to collect this client's traffic. In Figure 3.1, we depict a client located in AS X accessing two webpages hosted on an IP in AS W. The client traffic is represented by red lines. If the adversary controls AS X, they observe all the traffic related to the page visits. This is the adversary typically considered in the website fingerprinting literature [85, 61, 62, 29, 54, 19, 21, 55, 22, 88, 14]. If the adversary controls AS Y or AS Z, however, they would have limited visibility on the traffic, *i.e.*, they might not observe traffic from *all* clients' visited webpages, or for each observed web page, they might only observe a portion of the traffic (*e.g.*, the loading of some resources). We note that it is possible for an adversary to control multiple ASes or an IXP (where traffic from multiple ASes can traverse) [100, 101].

3.3.1 Website fingerprinting

As in previous work, we implement website fingerprinting attacks as a supervised learning problem. The adversary identifies the IP that contains the domains that they want to target. Then, the adversary enumerates all the domains on that IP, and collects web traffic traces from these domains. The adversary extracts features from these traces, and uses the feature vectors to train a classifier. Given a new trace, this classifier predicts which website it belongs to.

We implement the attack using a random forest classifier, a simple and effective model frequently used in website fingerprinting; and Var-CNN [88] to validate our results against the state of the art [14]. We use the set of features proposed by Hayes *et al.* [21] for performing website fingerprinting on Tor since this is a comprehensive set of features from previous related works. To adapt them to the QUIC case, we add features about packet size frequencies. Since QUIC's maximum payload size (1400 B) is smaller than that of TLS (16 kB), we compute frequencies of packet sizes up to 16 kB to encompass both QUIC and TLS traffic.

3.3.2 Adversarial Capabilities and Goals

We assume applications and websites hosted behind content delivery networks that use QUIC, and additional protocols, such as TLS ECH, to protect sensitive information. The information available to the adversary comprises the server's IP address, which is determined by the content delivery network, and any application-specific metadata such as the size of encrypted data, which is determined by the application. As the IP is known, an adversary can enumerate the domains this IP hosts (*e.g.*, through DNS reverse look-ups or from DNS scans using public name sources, including Certificate Transparency logs), collect pages associated with the domain, and train a classifier on these samples. This is

Table 3.1: Overview of datasets. All datasets except CHROMIUM are collected using Firefox.

Experiment	Identifier	# webpages	# samples
Landing pages (Mar'21)	MAIN	150	40
Influence of time (Sept'21)	TIME	145	40
Influence of client (May'22)	FIREFOX	131	40
Influence of client (May'22)	CHROMIUM	131	40
Domains (May'22)	HET	60	35

a much more tractable scenario than that of the open world Internet.

We study two adversarial models: unconstrained and constrained, depending on the adversary's visibility on traffic and the resources they can dedicate to fingerprinting. We also assume different adversarial goals and client setups that may impact the adversary's success. We summarize details of the datasets we collect to evaluate each of these scenarios in Table 3.1.

Unconstrained adversary.

An *unconstrained* adversary can observe and process *all* the traffic associated with a web page visit. We assume this adversary can have one of two goals:

Fingerprinting landing pages (Homogeneous Closed World). In this scenario we assume users only visit landing pages. Thus, the adversary only needs to collect landing pages to train their model. The training and testing set contain the same web pages, for which the traces vary due to content and timing changing when traces are collected at different points in time. As QUIC leaves IPs unprotected and the adversary can thus limit the anonymity set of a web page (see above) this corresponds to a classic *closed world* classification. This is a commonly used scenario in the literature [62, 21, 55, 22, 88].

Dataset creation Prior works rely on lists of most visited sites such as Alexa and Tranco to build datasets. These domains are typically hosted on different IPs, and thus would not be in the same anonymity set in our adversarial model. This would only happen if the client uses a VPN, which is out of the scope of this work. We work with a CDN provider to identify realistic anonymity sets that could be targeted by the adversary. We identify 13′744′979 unique domain names on the provider in Mar 2021. We filter out 2′641′100 errors (19.2%), 15973 NXDOMAINS (0.1%). Since we are interested in Domain Name → IP mappings, we ignore 1′024′234 CNAME answers (7.5%), and end up with 1′313′159 A/AAAA records. Finally, we use zdns in iterative to find these domains are hosted on 593′338 IPs, i.e., 593′338 anonymity sets for the adversary.

The anonymity set sizes follow the same distribution as that found by Patil *et al.* [102]

(Figure 3.2). We find that 60% of these domains are hosted on a unique IP address: when observing one of these IPs, the adversary is certain of which domain is being visited. Therefore, these domains are out of scope of our study. Only 50k IPs (8.5% of the dataset) host more than 150 domains, with one hosting as many as 56′319 domains. We choose a cluster of 150 websites for our experiments: which is a hard scenario among the 91.5% of the IPs served by the CDN provider. This cluster has a high percentage of TLS traffic: only 4% (std 1.7%) of the traffic is transmitted over QUIC. This results in the classifier focusing on TLS-specific features, preventing us from drawing meaningful conclusions about QUIC traffic's vulnerability or QUIC-oriented defenses (Figure 3.3). We also run experiments with other clusters of similar size and obtain comparable results.

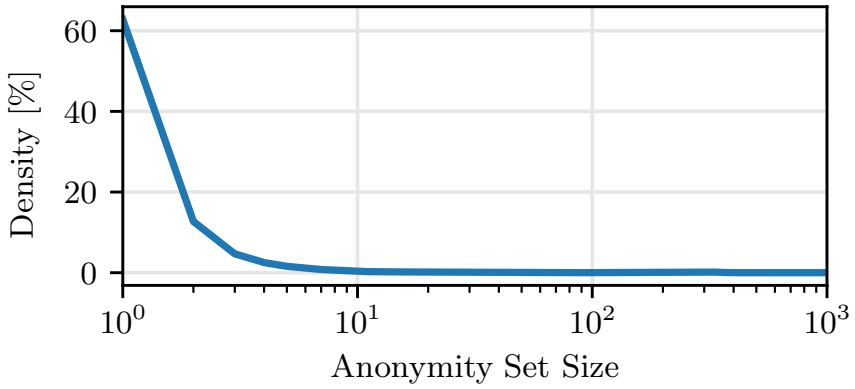


Figure 3.2: Distribution of the cluster sizes of 1.3M domains.

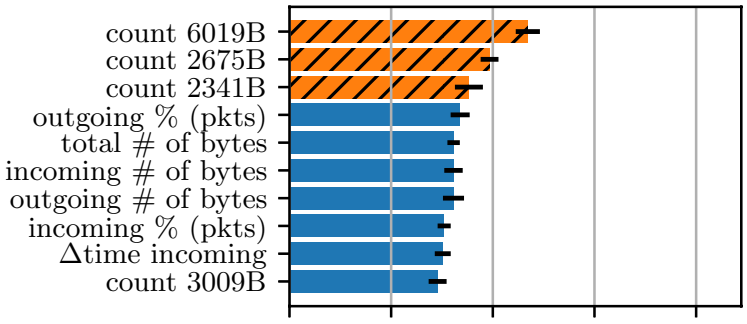


Figure 3.3: Feature importance for the cluster dataset. Due to low prevalence of QUIC, most of the features are TLS-specific (orange, dashed).

To address this problem, following the example of Smith *et al.* [14], we build QUIC-dominated datasets by crawling Alexa 1M [103], Umbrella 1M [104], and Majestic 1M [105]. We perform HAR captures[1] and we identify the protocols used by those websites. We

[1] A copy of the "Network" tab of the Firefox Developer Tools console.

select 150 domains that primarily use QUIC, and collect a dataset of traces for these domains, `MAIN` (collected in March 2021). `MAIN` has 70% of all traffic over QUIC (std 3%). To study the stability of website fingerprinting attacks over time, we collect one more QUIC-dominated set of traces from these domains in September 2021 (`TIME`).

Data collection. To build the datasets, we collect PCAP network traces from visits to each landing page of the domains in our list. We use Firefox 88.0 isolated in its own network namespace (using `netns`), enabling HTTP3, and disabling telemetry and auto-update settings to minimize extraneous traffic. We record 40 samples for each website. For each sample, we clean the caches by creating a fresh Firefox profile. We extract well-formed TLS and QUIC packets from the traces. To avoid relying on plaintext markers, we follow the approach of Smith*et al.* [14], and only extract the time and size of the sent and received packets.

In order to evaluate the influence of the client's browser, we collect `FIREFOX` and `CHROMIUM`– two datasets collected during the same time period, with different browsers: Firefox 98.0 and Chromium 101.0.

Fingerprinting domains (Heterogeneous Closed World). The previous scenario is somehow artificial in the sense that users visit more than the landing pages of domains. Thus, only training on landing pages would not work well in a realistic deployment. To model a domain, the adversary needs to train on both landing and subpages. Yet, due to the visited IPs being visible, the classification problem is still a closed world: the adversary has a finite set of domains to associate traffic traces to. We collect a dataset to evaluate how the adversary performs when also training on heterogeneous subpages of a domain.

Dataset creation We collect `HET` a dataset consisting of subpages for each domain hosted by the IP, instead of multiple samples of the landing page. We enumerate all pages that can be visited from the landing page (sharing the same domain as the landing page), and the pages that can further be visited from those sub-pages. Since the classifiers require a reasonable number of training samples, we limit our study with heterogeneous training set a to set of 60 domains hosted by the target IP which have at least 35 sub-pages. We collect this dataset using the process above with Firefox 98.0 as the browser.

Constrained adversary.

We assume that vantage points currently do *not* have the capability to run machine-learning tasks [106, 107]. They must mirror (part of) the traffic to a suitable location for analysis (purple dotted arrows in Figure 3.1). This location processes the traffic: it

extracts features and performs classification to identify the page visited by the client. In practice, mirroring all traffic is prohibitively expensive [31]. Thus, we also study a *constrained* adversary that only transmits summaries of locally computed statistics from sampled data [108, 109].

We measure the constrained adversary's cost to perform a website fingerprinting attack in terms of the bandwidth they require to collect and process the traffic traces. Bandwidth is a proxy for the required storage, as the adversary needs to store the transmitted information to query the machine learning model and possibly to retrain it. The computational cost is also proportional to the bandwidth, as the number and cost of operations needed to extract features depend on the length of the traces transmitted. We evaluate the adversary's success using filtered versions of the datasets described above.

3.4 Network-layer `PADDING`-based defenses

In this section we study whether building defenses using the `PADDING` frame at the network layer can thwart website fingerprinting attacks launched by both powerful and constrained adversaries.

3.4.1 Unconstrained Adversary

First, we evaluate the effectiveness of defenses against a powerful adversary that can observe all the traffic associated with a site visit, and can store and process all the traffic that it observes. Such an adversary could be, for instance AS X in Figure 3.1, if this AS would not have bandwidth or storage constraints.

We show the results for such an adversary in Table 3.2, for all datasets after 10-fold cross validation. The results are orders of magnitude better than random guessing (0.67%). We find that the random forest classifier provides a lower bound on the adversary's performance and gives us the advantage of interpretability, hence we use it for all our experiments. We also use Var-CNN [88] against the `MAIN` and `TIME` datasets to validate our results against the state of the art [14]. We obtain F1-scores of 92.28 and 94.22, showing that the adversary still has very high performance.

Unprotected traces.

Before studying defenses, we determine the performance of the adversary on unprotected QUIC traces on which no `PADDING` frames are used. We study different scenarios according to the goals defined in section 3.3.

Fingerprinting landing pages We first consider an adversary that is interested only in

Table 3.2: Mean classifier performance on full network traces for an unconstrained adversary.

Dataset	RF
MAIN	95.8 (std. dev. 0.4)
TIME	96.4 (std. dev. 0.2)
FIREFOX	95.6 (std. dev. 0.3)
CHROMIUM	92.9 (std. dev. 0.3)

landing pages, and assume that all users access the web using the same client (Firefox 88.0) In this case, for which the training and testing datasets contain only samples of landing pages, we observe the adversary obtains very good performance (F1-score above 90%). These results hold at any point in time: we observe very similar results for the TIME dataset. In all cases, the most important features are the histograms of packets between 1000 B and 1400 B, and the total transmitted volumes (Figure 3.4 shows importance for the case of MAIN).

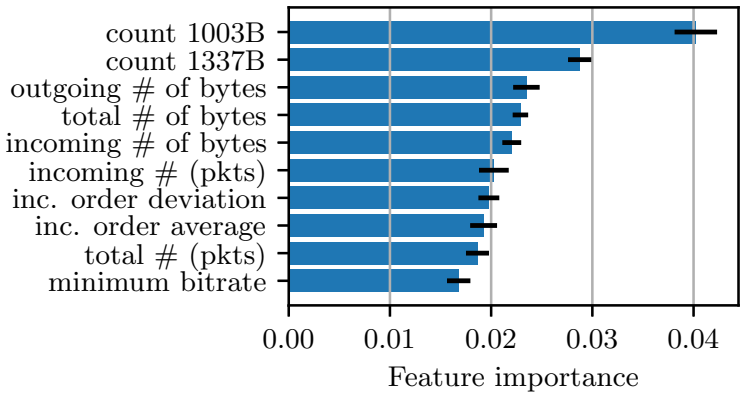


Figure 3.4: Feature importance for MAIN

Influence of time. The results in Table 3.2 assume that the adversary has collected the training set for their classifier close in time to their attack. This means that, as websites change over time, to obtain such results the adversary may have to repeat this collection if they are interested in launching the attack repeatedly. To understand the stability of the attack, we compare in Table 3.3 the performance of the attack when trained on traces collected at a time different than testing. We use the two datasets, MAIN and TIME, which we collected 6 months apart. We note that TIME has 145 domains instead of 150, due to failures in data collection (unreachable sites and repeated timeouts), hence we consider only these domains in our analysis. We observe that for both datasets, the classifier performance remains consistent when it is trained on data collected close to the attack. However, the performance drops significantly when used on data collected at a different

point in time. This indicates that for an adversary to be successful, they would need to update their training data with more recent samples to keep up with constantly evolving web pages.

Table 3.3: Influence of time. F1-score when training on the dataset indicated by the row and testing on the dataset in the column.

F1-score	MAIN	TIME
MAIN	95.8 (std. dev. 0.4)	35.2 (std. dev. 1.2)
TIME	17.5 (std. dev. 2.1)	96.4 (std. dev. 0.2)

Influence of user client. The previous experiments (using datasets (MAIN, TIME) assume that all users access the web using the same client and configuration. We now consider the influence of the client setup on the adversary's performance. We use two datasets, FIREFOX and CHROMIUM, collected during the same duration, for this evaluation. FIREFOX uses Firefox 98.0 and CHROMIUM uses Chromium 101.0. Both these datasets consist of 131 domains after accounting for collection failures. Table Table 3.4 shows that the classifier performs as expected when trained and tested on the same client setup. When the setup changes, we observe that the classifier performance drops, indicating that an adversary would need a classifier tailored to the client setup. Our observations are in line with those of [86]; we find that generally Firefox traces are longer, possibly due to more checks such as contacting OSCP servers.

Table 3.4: Influence of users' client. F1-score when training on the dataset indicated by the row and testing on the dataset in the column.

F1-score	FIREFOX	CHROMIUM
FIREFOX	95.6 (std. dev. 0.3)	35.6 (std. dev. 2.4)
CHROMIUM	22.9 (std. dev. 1.6)	92.9 (std. dev. 0.3)

Fingerprinting domains pages. Finally, we study the performance of an adversary interested in identifying the domain a user is visiting, regardless of the web page being visited within this domain. We use the FIREFOX and HET datasets, which use the same browser and are collected during the same time period. Table 3.5 shows the results. FIREFOX acts as our homogeneous closed world baseline (fingerprinting only landing pages) and is the first row in the table. For HET, we evaluate two cases. In the first case (second row of table), we assume that the adversary has seen all the subpages of a site, i.e., the training and test data are the same. In the second case (third row of table), we assume that the adversary has an incomplete set of subpages on which they have trained the classifier, and has to classify new subpages. We use 30 subpage samples per domain

to train the classifier, and 5 subpage samples to test (with different train and test samples for each fold). To provide an appropriate baseline, we use only the domains found in HET, and the same number of samples, for the FIREFOX dataset. We find that there is only a small drop in performance when we use subpages; using unknown subpages for the attack leads to largest drop of ≈ 3%. The adversary's success is most likely caused by subpages of the same site sharing many characteristics, and thus resources, leading to very similar traces. Our experiment indicates that the presence of subpages does not pose a large challenge for the adversary.

Table 3.5: Homogeneous vs Heterogeneous closed world experiment.

Scenario	Dataset	F1-score
Homogeneous (baseline)	FIREFOX	97.8 (std. dev. 0.4)
Heterogeneous (all subpages known)	HET	96.4 (std. dev. 0.9)
Heterogeneous (unknown subpages)	HET	94.1 (std. dev. 1.1)

PADDING-based network defenses.

To study whether defenses built on the QUIC PADDING frame can be effective, we focus on the worst case: fingerprinting full landing pages visited with the same client and with fresh training datasets (first row in Table 3.6). We run our experiments on the MAIN dataset.

Table 3.6: MAIN dataset: Mean classifier performance on defended traces.

Variant	F1 Score	Std. dev.
undefended	95.8	0.4
hiding individual sizes	93.9	0.4
hiding all timings	95.5	0.3
+ hiding total transmitted sizes	92.2	0.5

As we discuss in section 3.2, current defenses are oriented to protecting Tor traces, and none of the existing methods are directly applicable to QUIC traffic. Thus, we explore potential defense strategies that hide local and global features. We ignore practical considerations and assume that there exists an implementation that perfectly protects these features. Table 3.6 reports the results of the attack against difference defenses.

Hiding local features. The top predictive features for MAIN are all sized-based (Figure 3.4). We evaluate a defense that hides individual packet sizes. The attacker performance slightly decreases with respect to undefended traces. Padding individual packets poorly hides the total transmitted volumes, which becomes the top feature once individual sizes are removed. Additionally, hiding timings reduces the adversary's performance

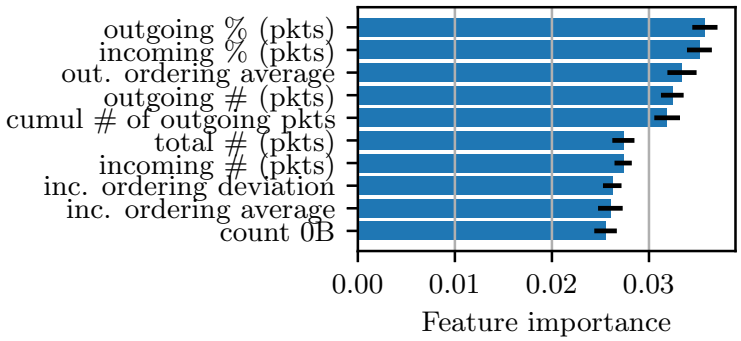


Figure 3.5: Feature importance when hiding hiding global features (last row of Table 3.6).

minimally.

Hiding global features. To hide the total transmitted volume, we increasing the size of all packets such that the total transmitted size is padded to the next megabyte. This yields another small drop in performance, as the attacker simply starts using packets orderings as main feature (Figure 3.5) which are almost as informative as sizes.

Injecting dummies. We then include dummy packets (padded to the maximum size, like the packets) to hide individual packet orderings. Since we cannot use existing defenses based on adversarial perturbation to find the optimal placement of dummies (see section 3.2), we inject dummy packets at random positions in the padded trace. We inject dummies up to a maximum of 60 seconds after the last real packet. This is a conservative value, as in our dataset the maximum duration for loading a website is 33s.

Figure 3.6 reports the effect of the proportion P of dummies injected in the trace on the adversary's F1-score. Dummies can only achieve a significant reduction at a sharp increase in cost: to obtain a $\approx 10\%$ F1-score drop for the adversary, we must add $+50\%$ bandwidth overhead. This is in addition to the already large overhead in terms of padding used to hide local and global features (mean cost of 612 kB per trace, with a large standard deviation: 440 kB).

3.4.2 Constrained Adversary

So far, we assumed the adversary can observe and process *all* of a victim's traffic. As described in section 3.3, there are on-path adversaries who might not observe all traffic from a particular client, nor all traffic to a particular server of interest; or that may not have the capability to process this traffic or to transmit it to a location suitable for analysis. We now study the performance of these adversaries, and whether `PADDING`-based

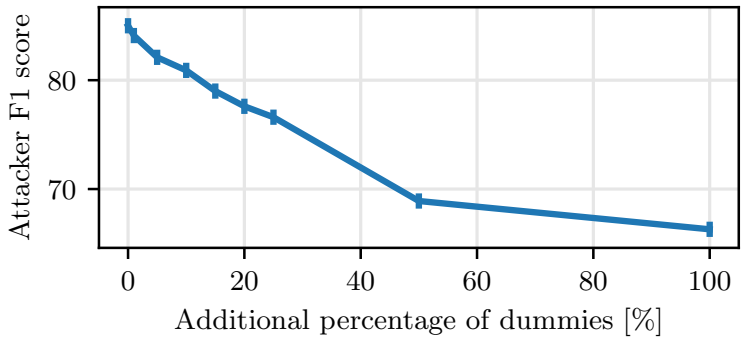


Figure 3.6: `MAIN` dataset: Mean classifier performance on traces with dummies.

network-layer defenses could protect against such adversaries. We run our experiments on the `MAIN` dataset.

Limited traffic visibility.

To understand the influence of limited visibility on the adversary's performance, we simulate an AS adversary with partial view on the client's traffic. In order to identify which parts of the traffic an AS would see, we use HAR captures to identify resources requested during page loads in our dataset. Then, as in prior work [110], we use traceroutes to record the path taken by all the resource requests, and the ASes encountered on each route. Concretely, we set `traIXroute` [111] to use `scamper` (configured with the Paris traceroute technique). As Juen *et al.* [110], we discard route hops that do not have IP or AS information (asterisks in the traceroute). To avoid inaccuracy in our analysis, we do not attempt to fill these gaps in routes via stitching. Thus, our results provide a lower bound on the amount of traffic that an AS adversary sees.

Once we know the routes, we simulate the partial view of the adversary by keeping only the packets associated to the resources visible from the adversarial AS vantage point: we filter out TLS/QUIC connections that do not correspond to the resources of collected through routes visible to the adversary. This filtering is based on the destination IP address, and the SNI when it exists. We observe a total of 974 routes in the `MAIN` dataset. These results are from traceroutes collected from the same location as the `MAIN` dataset (in France). We also ran traceroutes from other vantage points (in Germany, UK, Singapore), and observed the same trends. We report on these additional experiments in section B.1.

Figure 3.7 shows how many webpages are *seen* by each AS; we consider the web page is seen if the AS sees any traffic associated with this web page visits (including its

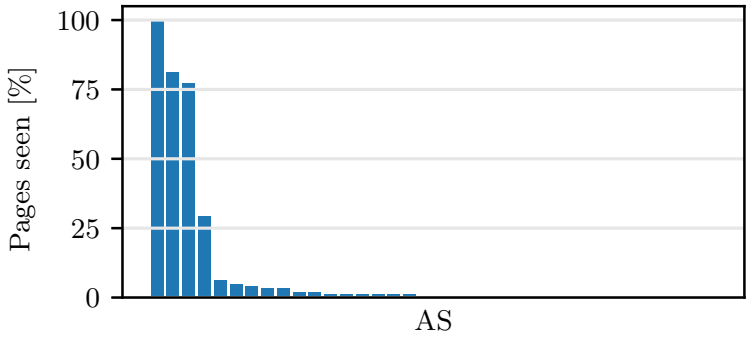


Figure 3.7: Distribution of web pages seen by each AS. Only three ASes (client's AS, Google, Cloudflare) can observe traffic from all the pages. Most ASes observe less than 10% of pages.

subresources). There are three ASes that observe traffic from more than 80% of webpages: one from Google, one from Cloudflare, and the AS where our client is located. The majority of the ASes, however, see only a small proportion of the sites (less than 10%). If these ASes were to be the adversary, they would not be able to fingerprint traffic from most websites hosted by the IP we choose as target for our attacks.

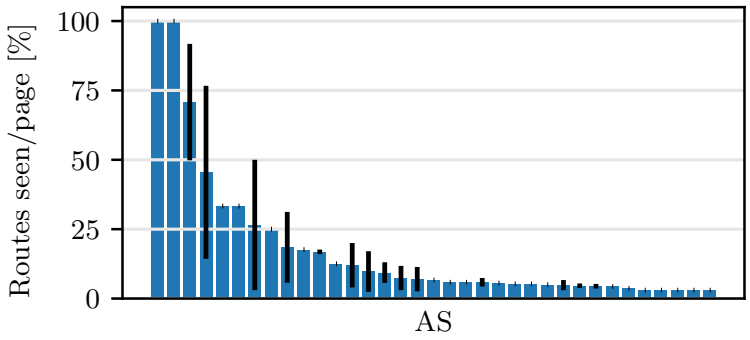


Figure 3.8: Distribution of routes per web page seen by each AS. Only three ASes (client's AS, OVHcloud, Google) observe more than 50% of the traffic per site.

To have a high attack success, an adversary must observe a sufficient proportion a web page traffic. For every web page an AS sees, we study what portion of this page they can observe (see Figure 3.8). As expected, our source AS sees 100% of the traffic of all pages in our dataset. Another AS, 4367 (belonging to OVHcloud), sees 100% of page traffic as well, but for a very limited number of pages. The Google AS is the second highest, seeing $\approx$ 70% of the routes for 80% of the pages in the dataset. All other ASes see less than 50% of the routes associated with the pages for which they can observe traffic.

We show in Table 3.7 the classifier performance for some ASes, if each of these ASes was the adversary. Few ASes have a substantial view of the client connections, *e.g.*, Google or Cloudflare. On the page loads they observe, these entities can fingerprint the traffic with high F1 score. Most other adversaries observe traffic from very few pages. Adversaries such as LEVEL3, Facebook, VNPT-AS-VN observe less than 5% of the pages in the dataset, meaning that they do not have any information to classify 95% of the pages. For the pages that they observe, they obtain high F1-scores. VNPT-AS-VN observes traffic for just one page and always identifies it, i.e., the anonymity set is 1. We conclude that, in order to successfully fingerprint, an AS adversary needs to observe a large proportion of the traffic, either by being the client's AS or by providing sub-resources on websites.

Table 3.7: Mean classifier performance on different AS views.

AS	Name	# Pages	F1 Score
15169	Google, LLC	118	89.5
13335	Cloudflare, Inc	115	92.9
3356	LEVEL3	7	81.7
32934	Facebook, Inc	5	92.3
45899	VNPT-AS-VN	1	100.0

Given these results, we do not expect different insights regarding the effectiveness of network-layer `PADDING`-based defenses in this scenario than when assuming an unconstrained adversary. For ASes with high visibility on the traces the results would be the same. For ASes that observe little traffic, the performance loss is already large. Thus the gain provided by defenses can only be marginal, while still imposing high bandwidth overhead.

Limited processing power.

To perform website fingerprinting, adversaries must have storage and computation capabilities, which middleboxes typically do not have. Mirroring the traffic to a location suitable for analysis requires resources. In fact, typical network monitoring solutions only record aggregate statistics over sampled traffic [112]. Common tools for network sampling include *NetFlow* and *sFlow* [100]. More efficient techniques have been proposed in academic papers (*e.g.*, sketching [113, 114] or *skampling* [115]), but to the best of our knowledge, they are not widely used. We focus on NetFlow, which is widely deployed on the Internet.

To investigate whether NetFlow statistics are sufficient to perform traffic-analysis attacks, we simulate *Sampled NetFlow*, a variation of NetFlow used in high-speed links where packets are first sampled in a deterministic fashion (1 out of every N packets) and

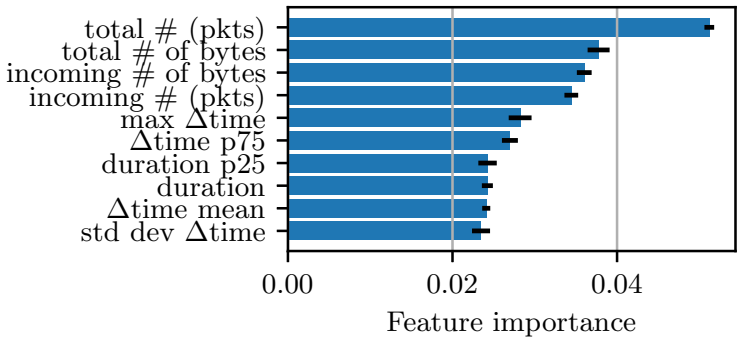


Figure 3.9: Feature importance for classifying NetFlow with 1% packet sampling rate.

flow statistics are computed on the sampled packets. First, we down-sample the PCAPs uniformly to the desired sampling rate, and we create NetFlow summaries from the PCAPs using `nfpcapd` and `nfdump`. We experiment with various sampling rates: 100%, 10%, 1% or 0.1% (common sampling rates in the wild range from 50% to 0.1% [109]). Second, we adapt the features used by our model to NetFlow summaries, which record the number of packets but not their sizes, timings, or directions, as follows: we consider a flow as a single packet whose size is the sum of all packets in a flow, and inter-packet timings become inter-*flow* timings. We acknowledge there could be better, tailored features and thus our evaluation only provides a lower-bound on the attacker performance.

Table 3.8: Mean classifier performance and median storage cost per sample for Sampled NetFlow, `MAIN`.

Sampling	**F1 Score**	**Size [kB]**
Full traces	95.8	312.4
NetFlow 100%	90.5	25.9
NetFlow 10%	66.4	3.0
NetFlow 1%	41.7	0.9
NetFlow 0.1%	16.8	0.4

We show the mean classifier performance on the NetFlow summaries in Table 3.8. As expected, moving from full packet data to flow summaries leads to a significant reduction in the adversary's performance: 29 percentage points lost when 10% of packets are sampled. Yet, the F1-score for any sampling rate is much higher than random guessing (F1-score=0.6%).

Defending NetFlow traces Regardless of the sampling rate, the most important features when using NetFlow are the total number of bytes and packets (Figure 3.9). We explore

Table 3.9: Mean attacker performance on defended NetFlow, `MAIN`.

Sampling	F1 Score
NetFlow 100%	53.1
NetFlow 10%	33.1
NetFlow 1%	21.6
NetFlow 0.1%	8.6

a defense that hides both per-flow metrics and overall statistics about the number of bytes and packets exchanged. For full traces, we hide global statistics by padding the total transmitted bytes to 22 MB and the number of packets to 25K. This padding is added uniformly to all the flows of one sample. These are the maximum transmitted size and number of packets we observe in our dataset. For sampled traces, the maximums diminishes linearly with the sampling rate. We apply the same reduction in the padding function.

The defense reduces the attacker performance, yet at an impractical cost; the median cost is ≈ 39 MB per trace (see Table 3.9). Most of the gain in privacy compared to the standard setting (95.8%), however, comes from the sampling rather than the defense (for example, (-54.1% via sampling vs. -20.1 % via the defense for Netflow 1%).

Inexpensive fingerprinting due to resource centralization.

In the previous two sections, we have shown that partial visibility and low storage and computation capabilities significantly hamper the adversary's ability to infer users' browsing patterns. We now show how the common use of Google resources by web developers can be used by constrained adversaries to bypass these limitations.

When looking at the HAR captures, we observe that the main reason for network traffic traversing Google's AS is that most websites request resources from a Google-owned domain (125 websites in `MAIN` (83% of the dataset)). We confirm this result in subsection 3.5.1.

While studying the traces, we also observe that the order in which these resources are loaded is website-particular, i.e., even when two sites load the same set of resources are loaded, these resources are loaded at different times (with respect to the time of query of the home page) and in different order. Such website-dependent behaviour can be seen as a fingerprint. Therefore, the centralization of resources can be used by an adversary to perform traffic analysis at a fraction of the typical cost: instead of recording all traffic, an adversary can use the timings of `ClientHello`'s to Google IPs to efficiently fingerprint

the traffic.

To validate this hypothesis, we study the performance of an adversary that only records the traffic towards Google services. We do this by filtering the network traces for which the destination IP (or the SNI) belongs to Google. If this field is not present in the packet, we perform a reverse-mapping with the destination IP to confirm the destination. To list domains belonging to Google, we get the requested URLs using our HAR capture, and we check the ownership using Tracker Radar [116]. In our attack, we use the following four Google-owned domains: `google.com`, `gstatic.com`, `youtube.com`, `doubleclick.com`, `ggpht.com`. Finally, we extract the sending times of the packets containing a `ClientHello` to these Google IPs and domains. For `MAIN`, this represents 7.6 floating-point values on average per loading of a website, with a maximum of 27 values. The size of the fingerprint is between 61B and 216B per loading of a website; in contrast, the mean PCAP size is 112 kB for the traffic towards Google, and 312 kB for all traffic.

Table 3.10: Mean classifier performance and median storage required per sample on traces filtered by connections to Google services, `MAIN`. The last two rows use 125 samples.

Variant	**F1 score** (Std dev)	**Size [kB]**
Baseline (Full Traffic)	95.8 (0.4)	312.4
Full traffic to Google	78.4 (0.7)	112.1
`ClientHello`'s to Google	66.1 (0.4)	0.1

We show in Table 3.10 that even *using only the timing of requests to Google*, the adversary achieves 77.9% F1 score for the 125 websites that use some Google resource. To obtain this result, the adversary needs *only* ≈ 61 *B per connection*, a saving of four to five orders of magnitude compared to recording full network traces. The feature analysis confirms that the timings between sub-resources is what helps the attacker in this case (Figure 3.10).

3.4.3 Take-aways

Our experiments confirm that website fingerprinting is a threat for QUIC traffic [14], not just when assuming that the adversary is only interested in landing pages, but also when fingerprinting subpages. While factors such as time and user client can influence the adversary's performance, these can be overcome by retraining the classifier with newer data, and by using classifiers tailored to specific client setups. We show that the threat persists even when the adversary has limited visibility or storage capabilities. Interestingly, the fact that resources are centralized in a few CDNs can enable other entities to perform fingerprinting at a fraction of the usual cost: *e.g.*, an ISP can use the Google-filter to

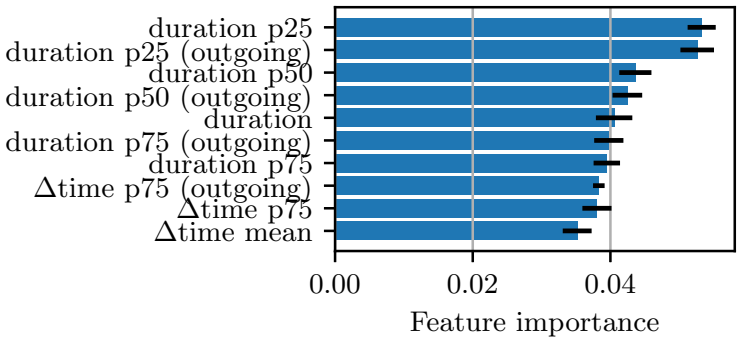


Figure 3.10: Feature importance for classifying websites based on the timings of their requests to Google services.

save bandwidth by 3 to 4 orders of magnitude (compared to running the attack on all traffic) and still efficiently fingerprint.

We then studied the potential of network-layer `PADDING`-based defenses to thwart website fingerprinting attacks. Against an unconstrained adversary, we find that, despite their high cost, these defenses can only reduce the adversary's success by, at most, 3.5 percentage points without dummies. Dummy injection has poor trade-offs: a 50% overhead to achieve a 10% performance reduction. Against a bandwidth-constraint adversaries, we find that network-layer `PADDING`-based defenses help, but the bulk of the privacy gain stems from the sampling rather than the padding.

The reason for the failure of network-layer `PADDING`-based defenses is that the anonymity sets behind IPs are usually small, and the classifier is able to pick even small differences between the traces. The defenses we study cannot hide these differences as *at the network layer, they lack information on the total transmitted size and the total number of packets that will be sent.*

3.5 Designing application-aware `PADDING`-based defenses

The results above confirm that the findings of Dyer *et al.* [59] for HTTP over encrypted tunnels hold for QUIC traffic: network-layer defenses have limited ability to hide global traffic patterns. To efficiently hide global traffic patterns (*e.g.*, total incoming size), a defense needs to know the size of the objects in advance. The only way to obtain this is to know the page's resources, which can only be obtained at the application layer.

In this section, first, we analyze the structure of websites to gain understanding about what information can be extracted and used to inform defenses. Second, we study different

ways in which this information can be used to build defenses, and whether such defenses fare better than their uninformed network-layer counterparts.

3.5.1 Understanding web page composition

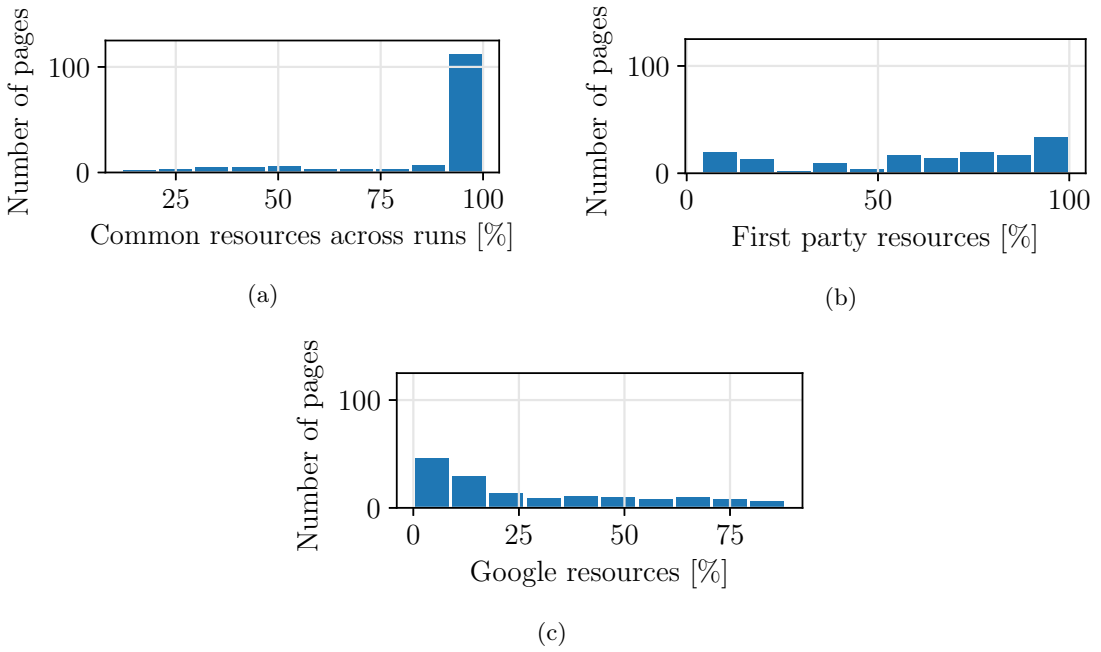


Figure 3.11: Resource dynamism and ownership for pages in the `MAIN` dataset. (a) Distribution of the proportion of resources that remain static across 5 runs. The majority of resources remain constant across runs, indicating low page dynamism. (b) Proportion of first party resources. 18% of the pages have less than 20% of first-party resources. (c) Proportion of Google third party resources. 24% of the pages have more than half their resources served by Google.

We study different dimensions related to the structure and composition of websites that are relevant for configuring `PADDING`-based defenses. For this purpose, we use OpenWPM [117] to analyze the webpages in the `MAIN` dataset. OpenWPM logs the HTTP requests that occurred during the page load. Unlike HAR captures, OpenWPM also records the originator of a request. We collect page structures by crawling the pages in `MAIN` with OpenWPM (v0.17.0) using Firefox five consecutive times.

Resource dynamism. We first study how pages vary across crawls, *i.e.*, how dynamic the pages in our dataset are. Dynamism influences how easy it is to protect a page. The less dynamic pages are, the easier it is to protect them as one can select defense parameters tailored to the static resources. If pages vary overnight, defenses can only be configured to fit the average case.

Out of the 150 websites in `MAIN`, 149 were successfully visited across all crawls. For these pages, we calculate the proportion of resources that remain static across the runs. Sometimes, even if the resources fetched are the same, the URL parameters may vary. Hence, we strip the URL parameters, and plot the distribution of static resources in Figure 3.11a. The mean proportion of static resources is 88.25% (Std: 22.46%) and the median is 100%. This indicates that pages in our dataset mostly contain static content. We note, however, that our measurements are taken over a short period of time and dynamism could become more prevalent if web pages are observed over a longer time period.

Resource ownership. Next, we study the ownership of these resources. Resource owners are the ones that can provide information about resources and modify them. Thus, understanding the variety of owners is important to get an idea of how much coordination among them would be required to protect a page from website fingerprinting.

We define ownership in terms of whether a resource is *first* party (shares the same eTLD+1 as the page) or *third* party (has a different eTLD+1 as the page). For example, on the page `www.example.com`, a resource `img.example.com` would be first-party and a resource `external.com` would be third-party. Using domains as a proxy for ownership is not perfectly accurate: content for `facebook.com` can be served from `fbcdn.net`, and both domains come under the control of Facebook, but the latter would be identified as a third party. Unfortunately, we could not use existing services that provide entity-domain mappings [116] as these services do not have relevant ownership information for almost half of the resources in our dataset. Thus, our ownership results are determined by domain name.

Figure 3.11b shows the proportion of first-party resources over the pages in our dataset. A majority of pages have a large proportion of first-party resources (Mean 61.18%, Std 31.61%, Median 66.67%). At the same time, the proportion of first-party resources can be as low as 3.92%. On average, there are 5.95 unique third party domains per page (Std: 7.64, Median: 3), with the number of domains going up to 44 for one of the pages in the set. This indicates that protecting all resources on a website would require coordination among multiple third parties to configure the `PADDING` frames in an efficient way. Visual inspection of the third-party domains shows a large number of domains commonly associated with Google. We map the domains to their owning entities [116] to measure Google's prevalence ([116] contains mappings for Google's domains). Figure 3.11c shows the proportion of Google resources per page. 24% of the pages have more than half their resources served by Google.

3.5.2 Application-aware defense strategies

Party-based resource protection.

The adversary can filter resources coming from different parties and perform the attack on traffic from different origins (like only on Google resources as in Table 3.4.2). Before diving into designing strategies, we assess whether third parties need to be involved or if only protecting first-party resources suffices.

We build traces with only resources of first or third parties, and we only protect those using `PADDING` frames. First-party protection represent a scenario where web pages protect their content using some defense, but third-parties do not cooperate. Third-party protection represent a scenario where third parties such as CDNs, which host a large number of resources, decide to implement a defense, but smaller first-parties do not. We assume the adversary attacks the remaining undefended traffic.

We report the adversary's performance in the different scenarios in Table 3.11 . We find that the adversary can achieve a high performance by just analyzing partial, undefended traces, regardless of the origin. Thus, for any defense to be effective, all parties serving content for a page must be coordinated and actively participate in the protection of resources. In particular, due to its prevalence, *Google must collaborate for any `PADDING`-based defenses to be efficient.*

Table 3.11: Mean classifier performance on traces filtered by 1st / 3rd party and Google CDN, `MAIN`.

Variant	F1 Score	Std. dev
All traffic	0.937	0.008
Only traffic to/from 1st parties	0.955	0.004
Only traffic to/from 3rd parties	0.915	0.005
Only traffic to/from Google CDN	0.914	0.006

Evaluating application-aware defense strategies

Application-aware defenses can be implemented both at the application layer or at the network layer (if information is passed to the middle-ware implementing padding). In both cases, the goal is to perturb features at the application layer. Thus, we directly evaluate perturbed application-layer traces. This gives an upper bound on the performance of a network-layer adversary with respect to a set of features [118]. The reason is that features at the network layer are effectively a noisy version of the resources at the application layer [118] (e.g., the number of incoming packets and the total size of incoming QUIC packets

are a noisy version of the actual size of the downloaded resources, and the total duration of the connection is a noisy version of the total amount of bytes downloaded).

We study three scenarios: undefended traces with all features, undefended traces without timings, and defended traces without timings. The latter is a good estimate of the attacker performance (even with timings), as our baseline analysis shows (see Figure 3.5.2). Since Var-CNN [88] relies on packet orderings and timings, but does not consider packet sizes, we limit our evaluation to the random forest classifier.

Metrics. We use two metrics to evaluate the defense's success: the performance of the classifier and the overhead imposed in terms of kilobytes of data added per subrequest.

Dataset and features. For the undefended baseline, we use the HAR captures of `MAIN`. We derive the k-Fingerprinting features these captures, which output a list of tuples $[t_{request}, size_{request}, t_{response}, size_{response}]$.

In practice, our padding and dummy-injection defenses would affect the timing of packets. However, without deploying the defenses we cannot predict these changes would reflect on our traffic captures. Deployment is not a possibility, as even if we would copy all websites of `MAIN` on a server we control, we would not be able to simulate actions from third parties. Fortunately, timings are less stable (and hence, less useful) than sizes, and therefore, they are not among the most important features (Figure 3.12). In fact, when attacking full HARs, the adversary obtains very good performance (93% F1 score) both with and without timing information. The most important features are size-related, being `bytes_outgoing` the most important feature by a slight margin over `bytes_incoming` (`bytes_total` is the sum of the two). This is corroborates the findings by Hentgen, who shows that even without timings evaluating at the application layer yields an upper bound over the network layer [118]. In the remaining experiments, we discard timings.

Defense strategies.

We now discuss possible strategies that use application-layer information to decide how to configure `PADDING` frames.

Protecting local features with padding. We design a padding function $\texttt{pad}_{\text{resources}}$ to hide individual queries and resources sizes. Such a defense must be implemented both on the client and the server. To configure this padding function, we use (1) the distribution of request sizes in the targeted set of websites and (2) a parameter N, which defines how many different sizes the defense allows for. The padding function to splits the resources sizes into N groups of equal density. For instance, if $N = 1$, all resources are padded

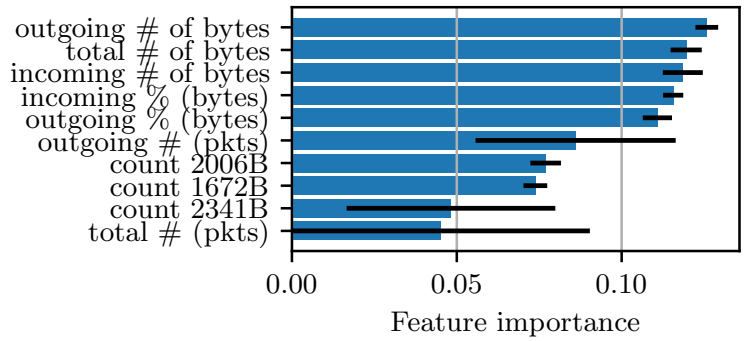


Figure 3.12: Feature importance for `MAIN` when using application-layer features (based on HAR captures).

to the max resource size in `MAIN`; and if $N = 2$, half of the resources are padded to the median size, half to the max size. Choosing a small N increases privacy: more resources will be padded to the same size and be indistinguishable; but also increases bandwidth usage.

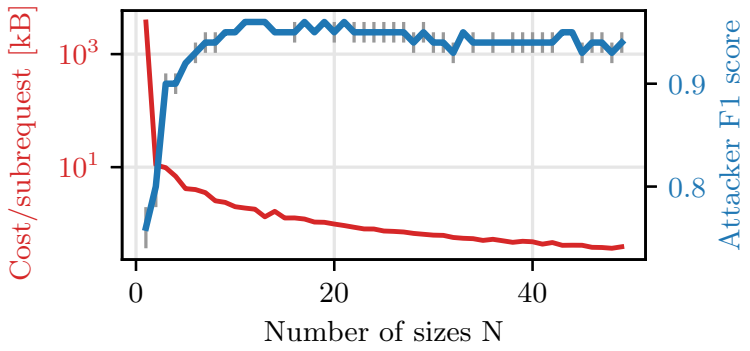


Figure 3.13: Number of allowed sizes, N, in $\texttt{pad}_{\text{resources}}$ versus attacker performance and median bandwidth cost per subrequest.

We run this defense varying N, and plot the median cost and the attacker F1 score in Figure 3.13. Only large amounts of padding (small N) have an effect on the attacker accuracy. Padding with large sizes has little effect. For instance, $N = 3$, which results on packets of 5.58 kB, 21 kB, 3.6 MB, decreases the accuracy adversary by 6% and incurs a median overhead of 9 kB per resource. This ineffectiveness stems from the fact that the adversary still has access to the number of requests and overall volume (Figure 3.14), which are sufficient for the attack. The traces' total size are too different to be efficiently hidden through the padding of individual resources.

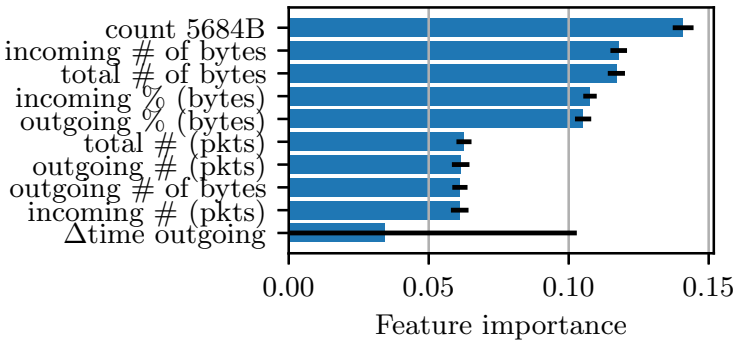


Figure 3.14: Feature importance with 3 padding sizes: 5.58 kB, 21 kB, 3.6 MB.

Protecting global features with padding. Padding only individual packets' size cannot protect the overall transmitted volume. We design a padding function $\texttt{pad}_{\text{total size}}$ to pad the total incoming and outgoing traffic sizes. To evaluate the best case defense, we assume the ideal scenario in which the padding effort is split evenly across all the parties queried on one web page. This way, the adversary does not gain an advantage by filtering the traces from one party in particular. This strategy assumes the existence of a mechanism by which clients can ask third parties for a particular amount of padding per resource; how to design such a mechanism is outside the scope of this work.

The defense has one parameter, N, which defines how many total incoming and outgoing traffic sizes are allowed. We first compute the maximum total incoming and outgoing traffic in our target dataset, `MAIN`. The maximum total size of queries in one website is 102 kB and the median is 14.4 kB; and the maximum total size of all downloaded resources is 8.19 MB, with median 750 kB. To apply the defense, we split the total incoming/outgoing sizes into N groups of traces with equal density. For instance, when $N = 1$, there is only one group of maximum size: all websites' outgoing traffic would be padded to 102 kB, and the incoming traffic to 8.19 MB. For $N = 2$, the groups would correspond to the median and to the maximum total incoming and outgoing traffic. For $N = 3$, the groups would correspond to tertiles of the distribution, for $N = 4$ to quartiles, and so forth. We allocate every website to the group that is closest to its original total incoming and outgoing size, and we spread the padding evenly across all queries and resources of that website.

We run this defense varying N, and plot the median cost and the attacker F1 score in Figure 3.15. As in the network layer, padding the total size does not mitigate the attack. For instance, to drop the adversary's accuracy by 10 percentage points, the defense incurs a median cost of 5.7 kB per request (outgoing traffic) and 300 kB per resource (incoming traffic). In the best case, it reduces the attacker's accuracy by 16 percentage

point, with a median cost of 109 kB per request and 8.16 MB per resource.

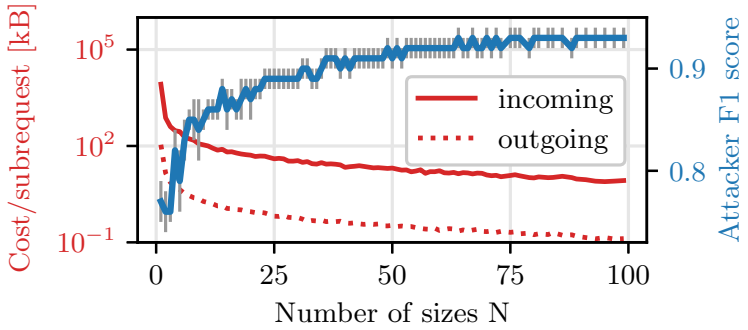


Figure 3.15: Number of allowed sizes N in $\texttt{pad}_{\text{total size}}$ versus attacker performance and median bandwidth cost per subrequest.

Protecting global features with dummies. An orthogonal, popular approach to hiding the total size of a web page is injecting dummy traffic [70]. Unlike in Tor, where dummies are indistinguishable from real cells due to the standard cell size, in QUIC, care must be taken that dummies' sizes do not enable the adversary to filter them. To evaluate the defense in ideal circumstances, we assume the existence of a dataset that contains the most popular queries and resources across all web pages to be defended. Then, we build dummies replicating the structure of those popular queries. This ensures that the queries are hard to filter for an adversary.

In our experiments, we select popular resources from Google (fonts, analytics, static assets). When a web page is loaded, we choose a number of resources to inject (M). These resources can themselves trigger additional queries. We flip a coin and with probability p, we inject a dummy resource. We inject these resources at random times over the duration of the connection, such that the adversary cannot use timing to identify and filter out dummies.

We plot in Figure 3.16 the attacker F1 score for a varying number of dummies. This defense is more effective than the previous ones. For instance, with $(p = 0.5, M = 10)$, which injects on average 5 dummy requests, the attacker's F1 score decreases from 93% to 54%, at a median cost of 137 kB per loading of a web page. In general increasing p has better impact on the attacker's performance than increasing the quantity of dummies (M): on average, the two parameters $(p = 0.5, M = 10)$ and $(p = 1, M = 5)$ inject 5 sequence of queries to a CDN; but the former reduces the attacker's F1 score to 0.54, and the latter to 0.43.

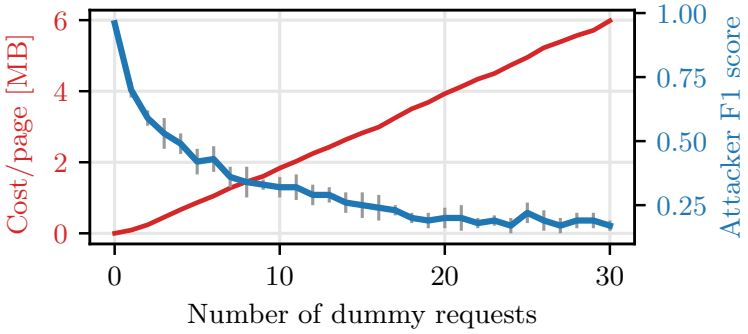


Figure 3.16: Attacker performance and cost when varying the number of dummies.

3.5.3 Take-aways.

As in the network-layer, padding resources individually or globally at the application layer incurs in large overhead and does not necessarily eliminate website fingerprinting attacks. Dummy injection, on the other hand, shows better trade-offs.

On the downside, contrary to pure network-layer defenses, when information is protected at the application-layer, there is a need for coordination between first and third parties. Moreover, to work properly, application-aware defenses need a-priori knowledge of the resources sizes and ordering. This may be hard to obtain when browser & website optimizations (*e.g.*,, client caching or pipelining of resources) are in place. Thus, the deployment of these defenses may require the collaboration of web pages and web browsers developers besides the coordination of parties that serve web content.

3.6 Discussion and Recommendations

Our work highlights the difficulties in developing an effective website fingerprinting defense for the Web. We carried out a comprehensive study of the ecosystem in which defenses are to be deployed, and provided evidence for the fundamental incompatibilities between today's Web communications and user privacy.

Network-layer challenges. First, we confirm that previous results which point out that network-layer defenses are not effective against website fingerprinting [59] also apply when the transport protocol changes from TCP to QUIC. The main problem stems notably from the differences in the total sizes of websites and result in identifying features [21, 69]. Hiding the total size is hard at the network layer, where the size of objects is not known in advance. Thus, without coordination with the application layer, a generic, application-agnostic defense using the QUIC's `PADDING` frame is an inadequate mitigation against

traffic-analysis attacks. Effective mitigations require application involvement, either in the application code or as part of the browser's functionalities.

App-layer challenges & Next steps. While defenses at the application layer can obtain better protection at a smaller cost, our investigation shows that the current Web development practices hinder the effective deployment of any defense. For instance, a straightforward defense would be to pad the total size of websites to a standard set of sizes. However, websites use resources hosted on different servers, and defenses must cover *all* resources to be effective (see subsection 3.5.2). The widespread use of third party resources means that achieving full coverage requires coordination among many different entities, which seems unlikely to happen organically. Without coordination between first and third parties, Web-oriented standard bodies (*e.g.*, W3C) and browser vendors could develop standard mechanisms for normalizing how third-party resources are requested and served to reduce the threat of traffic analysis. For instance, the definition of standard sizes for third-party served resources, and methods to request these resources such that all websites use the same order.

A solution to avoid coordination would be to rethink the trend of creating web development resources as a service, and go back to having first parties hosting and serving the resources. In this spirit, initiatives such as Web Bundles [119] (though raising other privacy concerns) would remove the need for coordinating between third parties, simplifying the implementation of defenses, and removing vantage points.

IPs & Anonymity set sizes. In the QUIC setting, the adversary is largely helped by the IPs addresses; they can be used to turn the website fingerprinting problem into a closed-world classification problem, to dissect traffic based on first and third parties, and to link together a client's packets.

To address the easy linking of packets, clients could use techniques such as MIMIQ [120] to leverage QUIC's connection migration feature to change their IP address; or Near-Path NAT [121] or MASQUE [122] to completely hide their IPs; or CoMPS [123] to hide IPs and split traffic across multiple paths. This would force the adversary to probabilistically stitch packets together to form traces. If the source/destination IP/port are not identifying one client, simply rotating QUIC's connection ID might also prevent the adversary from linking together one client's packets.

To address the ease of separating traffic, CDNs that also host websites (*e.g.*, Cloudflare) could proxy the traffic to third parties, such that all traffic is served from a single IP. Finally, the closed-world size could be increased by co-hosting multiple websites on one server, or making a larger number of websites available behind load balancers; or even

moved to open world if all web traffic would be downloaded via anonymous communication networks (*e.g.*, Tor [124]) or VPNs.

Part II

Metadata analysis for web privacy

4 WebGraph: Capturing Advertising and Tracking Information Flows for Robust Blocking

This chapter is based on the article:

"WebGraph: Capturing Advertising and Tracking Information Flows for Robust Blocking.", S. Siby, U. Iqbal, S. Englehardt, Z. Shafiq, C. Troncoso, in *USENIX Security*, 2022.

4.1 Introduction

Users rely on privacy-enhancing blocking tools to protect themselves from online advertising and tracking. Many of these tools—including uBlock Origin [125], Ghostery [126], Firefox [127, 128], Edge [129], and Brave [130]—rely on manually curated filter lists [131, 132, 133] to block advertising and tracking. The research community is developing machine learning (ML) approaches to automate the detection of advertising and tracking and make filter lists more comprehensive. The first generation of ML-based blocking approaches analyze network requests [134, 135, 136] or JavaScript code [137, 138, 139] to learn distinctive behaviors of advertising and tracking. However, these approaches are highly susceptible to adversarial evasion techniques that are already found in the wild, including URL and code obfuscation [140, 141]. The next generation of ML-based blocking approaches leverage cross-layer graph information from multiple layers of the web stack [142, 143]. These approaches claim better robustness to evasion than single-layer approaches, due to their use of *structural* features (the hierarchy of resource inclusions) in addition to traditional *content* features (the resource's network location or response content).

In this chapter, we show that state-of-the-art ad and tracker blocking approaches, such as AdGraph [142], are susceptible to adversarial evasions due to their disproportionate

reliance on easy-to-manipulate content features. We show that a third-party adversary can achieve 8% evasion success by manipulating URLs of its resources. Worse yet, an adversary can achieve near-perfect evasion—as high as a 96% success rate—if they collude with the first party, e.g, by using the CNAME cloaking technique already deployed by some trackers [144, 145].

We introduce WebGraph, the first ML-based ad and tracker blocking approach that *does not rely on content features*. WebGraph improves the cross-layer graph representation by capturing a fundamental property of advertising and tracking services (ATS): the flow of information from one entity to the browser's storage, the network, and other entities loaded on a page. The intuition behind WebGraph is to focus on the *actions* of the advertising and tracking services, rather than the *contents* of their resources. We posit that actions are harder to obfuscate. Advertising and tracking scripts need to store identifiers for users, and those identifiers must be shared with any other entity with which they wish to share data (e.g., via cookie syncing [146]). Thus, we build a graph representation of the page load by monitoring network requests, JavaScript execution, HTML element creations, and browser storage access. From this graph we extract *flow* features, which explicitly capture distinctive information flows in advertising and tracking. Our evaluation shows that WebGraph's graph representation and flow features can supplant content features, with comparable accuracy.

While high accuracy is necessary for deployment, it is not sufficient. We have repeatedly seen that advertisers and trackers will attempt to circumvent detection and evade blocking [140, 141, 144, 147]. Therefore, in order for an advertising and tracking classifier to be useful in practice, it must be robust to adversarial manipulation. We show that WebGraph represents a significant step forward in robustness to adversarial evasion as compared to prior approaches. In particular, we find that WebGraph is robust to the types of URL, CNAME, and content manipulation evasion techniques that are currently deployed on the web. We also know that advertisers and trackers will attempt to deploy more sophisticated evasion techniques tailored to our proposed approach. To understand how robust WebGraph would be in the face of these new evasion techniques, we propose a novel realistic graph manipulation evasion technique. We show that this attack achieves only limited evasion success against WebGraph, while incurring a non-trivial usability loss in terms of mistakenly blocking its own advertising/tracking resources or other benign resources on the web page.

Overall, our findings suggest that the community should migrate away from unreliable content features for advertising and tracking blocking. We show that information flow features built upon the actions of advertisers and trackers provide a promising path forward.

In summary, our contributions are as follows:

- We show that existing ML-based ad and tracker blocking approaches are susceptible to evasion due to their reliance on content features. As a representative example, we show how an adversary can achieve near-perfect evasion of AdGraph using techniques already in use on the web today.
- We introduce WebGraph, the first ML-based ad and tracker blocking approach that does not rely on content features and captures fundamentally distinctive information flows in advertising and tracking.
- Our in-depth evaluation shows that WebGraph achieves comparable accuracy to prior approaches and achieves significantly better robustness to adversarial manipulation of content features.
- We propose a novel graph manipulation evasion technique, and show that WebGraph (and the information flow features it relies on) remains robust under this attack.

Chapter organization: The rest of this chapter is organized as follows: Section 4.2 provides an overview of the recent advances in ML-based ad and tracker blocking. Section 4.3 evaluates robustness of existing graph-based approaches, using AdGraph as a representative example. Section 4.4 describes the design and evaluation of WebGraph. Section 4.5 further evaluates WebGraph's robustness to adversarial attacks. We discuss limitations of our work in Section 4.6 and conclude in Section 4.7.

4.2 Background & Related Work

Online behavioral advertising enables ad targeting based on users' interests and behaviors. To target ads, online advertising relies on the intertwined tracking ecosystem that uses cookies for cross-site tracking. For instance, the real-time bidding (RTB) protocol that powers programmatic online advertising has built-in mechanisms for advertisers and trackers to share information [146, 148, 149]. Thus, almost always, ads and trackers go together, with intertwined execution flows and resource dependencies. Below, we revisit prior literature on ad and tracker blocking, and analyze its limitations.

Popular ad and tracker blocking tools such as Adblock Plus [150] rely on filter lists [131, 132]. These filter lists are manually curated based on user feedback. Prior work has shown that manually curated filter lists suffer from *scalability* and *robustness* issues. First, filter lists have trouble keeping up with the ever expanding advertising and tracking ecosystem. Filter lists have grown to include tens of thousands of rules that are often not updated in a timely fashion. For instance, filter lists may take as long as 3 months to add rules for newly discovered ads and trackers [151]. Once a filter rule is added to block an

advertising and tracking service, it is rarely removed, even if it is no longer needed. In fact, prior work showed that almost 90% of the rules in filter lists are rarely or never used [140]. Second, filter lists are not robust to evasion attempts by advertisers and trackers. Filter lists are brittle in the face of domain rotation [152, 153] and manipulation of page structure [154, 155, 156]. For instance, prior work showed that filter lists are susceptible to evasion attacks such as randomization of URL path, hostname, or element attributes and IDs [157, 158, 147].

Addressing scalability. To address the scalability issues that arise due to manual curation of filter lists, researchers have proposed to use machine learning (ML) for automated ad and tracker blocking. Prior ML-based approaches mainly detect ads and trackers at the network and JavaScript layers of the web stack. These approaches detect ads and trackers by featurizing network requests [134, 135, 136] or JavaScript code [137, 138, 139].

Network layer approaches rely on content in URLs, HTTP headers, and request and response payloads (e.g., keywords, query strings, payload size) to extract features and train ML models to detect ads and trackers [134, 135]. While trying to mimic filter lists by detecting ad and tracker URLs, these approaches end up replicating some characteristics of filter lists and thus also naturally inherit their shortcomings. For example, presence of a certain keyword in the request URL could be a distinguishing feature. However, as discussed earlier, such keyword based features are brittle in the face of trivial evasions such as domain rotation [157, 158].

JavaScript layer approaches rely on static or dynamic analysis to extract features and train ML models to detect ads and trackers. Examples of features are n-grams of code statements obtained via static analysis [139] or JavaScript API invocations captured via dynamic analysis [137]. These approaches are susceptible to JavaScript obfuscation [159, 160, 161]. These approaches are also susceptible to evasion such as script amalgamation or dispersion. They implicitly assume that tracking code is bundled in a single script or that tracking scripts only contain tracking code. However, in practice, tracking code could be distributed across several chunks and packaged with functional code [142, 162].

Addressing robustness. While network and JavaScript layer approaches consider information at each layer in isolation, ads and trackers rely on all three layers (i.e. network, JavaScript, and HTML) of the web stack for their execution. Therefore, focusing on only one layer lacks robustness against the aforementioned evasion attempts. To address this limitation, graph-based approaches aim to capture the interactions among and across layers of the web stack.

Graph-based approaches extract features from the cross-layer graph representation to train ML models to detect ads and trackers [142, 143]. These approaches leverage rich cross-layer context and thus claim to be robust to evasion attempts. AdGraph was the first graph-based approach to ad and tracker classification [142]. It extracts structural features from the graph such as node connectivity and ancestry information as well as content features such as URL length and presence/absence of certain keywords. Sjösten *et al.* [143] introduced PageGraph, which extends AdGraph's graph representation by improving event attribution and capturing more behaviors. In addition to content and structural features, they also added perceptual features to train the classifier. Since *perceptual* features attempt to use the rendered resource content, they are also considered content features. Chen *et al.* [163] proposed an approach, using PageGraph, to detect trackers based on their execution signatures. In contrast to ML-based approaches, their signature-based approach would only be able to detect trackers that strictly match the signatures of tracking scripts, but miss trackers with even slight deviations in their behavior, such as changes in the execution order. Kargaran *et al.* [164] followed a different approach. Instead of building a graph representation per website, they combined graph representations across multiple websites to model relations between third parties on those sites. Just like AdGraph, they also extract structural and content features from the graph to train the classifier.

These graph-based systems use a combination of content and structural features for classification, which they claim increases the robustness to evasion attacks. While this combination should intuitively improve classifier robustness, we posit that it would be less robust than expected if the classifier relies heavily on content features. This is because content features pertain to a single node on the graph and are easy to manipulate for an adversary, e.g., using adversarial attacks on textual [165] and perceptual [166] content features, without causing undesired changes in other nodes. It is noteworthy that Zhu *et al.* [165], also manipulate structural features, however their manipulations are only limited to graph size. Further, they do not evaluate the impact of their mutations on overall graph.

In the next section, we analyze the robustness of graph-based ad and tracker detection systems. We focus on AdGraph as it is representative of other graph-based systems that use similar structural and content features.

4.3 AdGraph Robustness

In this section, we analyze AdGraph's robustness by evaluating its accuracy in the face of adversarial content manipulation.

AdGraph is a graph-based machine learning approach that detects ads and trackers based on their structural and content properties. AdGraph instruments the Chromium web browser to capture detailed execution of ads and trackers across the HTML, JavaScript, and the network layer, and models the interaction among these layers in the form of a graph. Using this graph, AdGraph extracts two categories of features: *content* (information related to individual nodes in the graph, such as URL length and presence of ad/tracking keywords in the URL) and *structure* (information about relationships between nodes, such as connectivity and ancestry information). It uses the extracted features to train a machine learning classifier to detect advertising and tracking resources. The full list of AdGraph features are described in Table C.1.

Since AdGraph relies on content properties, in addition to structural properties, it is subject to same evasion attacks that succeed against the filter lists-based ad and tracker detection approaches [157, 158].

4.3.1 Threat Model & Attack

Our threat model assumes an adversarial third-party advertiser or tracker embedded on a site, who aims to change the classification of its resources from advertising and tracking services (ATS) to benign resources (Non-ATS) in order to evade detection by ad and tracker blocking tools.

We assume that the adversarial third party has limited cooperation with the first-party publisher. We do not assume full cooperation because the parties are mutually distrusting. The third-party adversary generally does not trust the first-party publisher to serve its advertising and tracking resources via a reverse proxy [167, 168]. Likewise, the first-party publisher does not trust the third-party adversary to host functional resources via the adversary-controlled CDN [169]. Given existing practices, we assume that the adversary can serve its advertising and tracking resources from a first-party subdomain but not arbitrarily within the first-party domain space. For example, the adversary can masquerade its resources through CNAME cloaking [170], which only requires a minor change in DNS records by the first party. Recent measurement studies have reported an increase in the prevalence of CNAME cloaking over the last few years. Dao *et al.* [144] showed that the usage of CNAME cloaking-based tracking has steadily increased between 2016 and 2020, with 1,762 of Alexa's top-300K websites employing at least one CNAME-based tracker as of January 2020. Dimova *et al.* [145] also showed that the usage of CNAME cloaking has increased by 22% from 2018 to 2020, with 9.98% of Tranco's top-10K websites now employing at least one CNAME-based tracker as of October 2020.

We assume that the adversary is able to manipulate their own URLs by altering the domain name or query string. The adversary can only manipulate URLs that are under their control, and only attempts to manipulate URLs that were initially correctly classified as ATS (ad and tracker URLs initially classified as Non-ATS already benefit the adversary). The adversary cannot manipulate the data used to train the classifier. Therefore, we only implement mutations during inference.

We implement two types of URL manipulations. For domain names, we allow the adversary to randomly change the URL's domain, subdomain, or both. In practice, adversaries can rely on automated techniques to generate random domains and subdomains. For example, they can use malware-inspired domain generation algorithms (DGA) techniques to generate a large number of domains [152, 171]. For query strings, we randomly change the number of parameters, the parameter names, the parameter values in the URL, or a combination of the three.

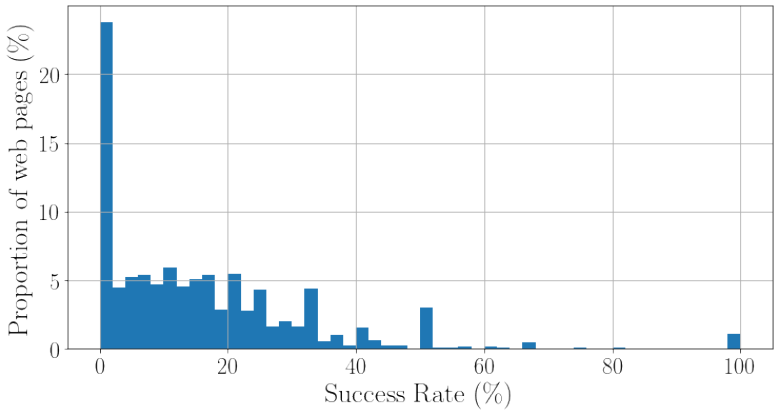


Figure 4.1: Classification switch success rate distribution by web page (over 10 folds) when the adversary does **not** collude with the first party. The average success rate per web page is 15.92 ± 0.03 %.

4.3.2 Results

Experimental setup. We extend OpenWPM [117] to automatically crawl websites with Firefox and build AdGraph's representation. We crawl 10K sites sampled from the Alexa's top-100K list, the top 1K sites and a random sample of 9K sites ranked between 1K-100K, and store their graph representations. Next, we implement a decision tree classifier that closely follows AdGraph's design [142], and extract features from the graphs for training and testing. For ground truth, we use the same set of filter lists for data labeling that were used by AdGraph [142]. A URL is labeled as ATS if it is present in one or more of the filter lists, and Non-ATS otherwise. We use 10-fold cross validation to obtain our results, where the folds are selected such that every fold uses a different set of web pages in the test set. Our classifier obtains comparable performance

to the original results reported by [142]: 92.33% accuracy, 88.91% precision, and 92.14% recall. The minor differences are likely due to differences in crawled sites, updated filter lists, and a few subtle changes in our adaptation of AdGraph from online to offline. In AdGraph's online implementation, features are extracted from each node in the graph as they are created. Our offline adaptation, instead, extracts features after page load completion. There are also some minor differences due to JavaScript attribution, caused by the differences in instrumentation between Chromium-based AdGraph and Firefox-based OpenWPM.[1]

Adversarial success rate without collusion. In our first experiment, we assume that the adversary does not collude with the first party. The adversary can randomize their domain and subdomain, but cannot masquerade as the first party. Our content mutation procedure results in the mutation of 41.48 ± 1.47 % of all the test data URLs (averaged over 10 folds). The adversary's success rate in evading the classifier is 8.72 ± 0.42 % (over 10 folds). While this may seem like a low percentage, we note that every successful mutation is a win for the adversary since it means that one more of their ads or trackers is now unblocked. Over all 10 folds, the adversary mutated 691,602 URLs, out of which 60,270 had their classifications switched.

We also observe that the evasion success rate varies across sites, as shown in Figure 4.1. For ≈1% of the web pages in the test set (90 pages), the adversary achieves a perfect success rate, meaning that all third-party ads and trackers on the web page are now classified as benign content. It is noteworthy that 21.62% of the unblocked URLs belong to popular ad exchanges, which are responsible for further diffusion of user information due to the broadcast nature of real-time bidding (RTB) [72]. These unblocked ad exchanges can amplify the privacy harm because they often share information about page visits with multiple advertisers and trackers.

Adversarial success rate with collusion. In our second experiment, we assume that the adversary colludes with the first party. The adversary can perform domain mutation such that their URL is a subdomain of the first party. The adversary's success rate increases to 96.62 ± 0.37 % (over 10 folds). This means that being able to use a first-party subdomain provides almost perfect evasion capabilities. Figure 4.2 shows the evasion success rate variation across sites. For ≈50% of the web pages in the test set, the adversary achieves a perfect success rate. We also see a higher proportion (32.25%) of the unblocked URLs belonging to popular ad exchanges, as compared to the previous

[1]Due to these differences, our features are not exactly identical to the online implementation of AdGraph. For example, in AdGraph, a node can have a maximum of two parents, which need not be the case for our system. Therefore, we do not use AdGraph features specific to these two parents. The full feature list, showing these differences is provided in Appendix C.1.

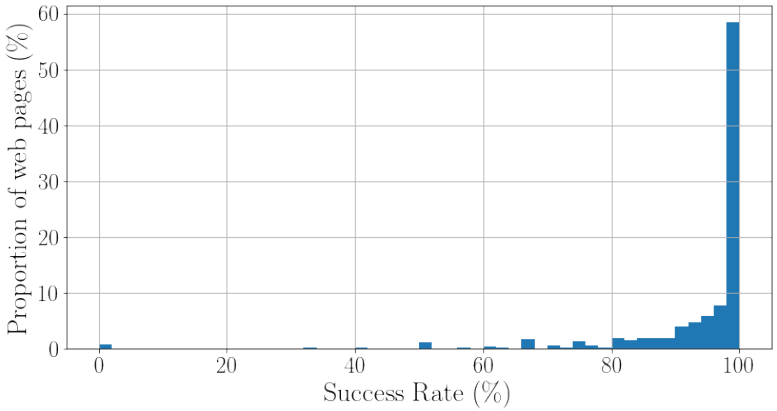


Figure 4.2: Classification switch success rate distribution by web page (over 10 folds) when the adversary colludes with the first party. The average success rate per web page is 93.01 ± 0.01 %.

experiment.

To better understand why such URL manipulation is able to evade detection by AdGraph, we analyze feature importance using information gain (see Table 4.1). We see that content features are essential to the AdGraph classifier: not only are the top-3 most important features content features, their relative importance scores are also high compared to the other features. Two of the top-3 features depend on whether a URL is third-party, which explains why we obtain high success rates when the adversary has the capability to masquerade as the first party. These two features do not have an effect in the case where the adversary does not collude with the first party, since the adversary cannot change the fact that they are third party. However, the adversary's manipulations still influence the third top feature, length of the URL. Hence, we observe lower but non-trivial success rates even without collusion.

Feature	**Category**	**Information Gain (%)**
URL length	Content	14.87 ± 0.36
URL domain is a subdomain of the first party	Content	11.06 ± 1.24
URL is a third party	Content	10.67 ± 1.32
Degree of a node	Structure	7.56 ± 0.63
Number of edges divided by number of nodes	Structure	7.48 ± 0.41

Table 4.1: Top 5 most important features for AdGraph's classification, their category, and information gain values (averaged over 10 folds).

These results show that graph-based ML classifiers such as AdGraph over-rely on content features that makes them vulnerable. Next, we propose an approach to improve the robustness of graph-based ad and tracker blocking tools.

4.4 WebGraph

Online advertising and tracking fundamentally relies on information sharing. Trackers need to share information with each other to improve their coverage of users' browsing history [117, 146]. Trackers also need to share information with each other as part of built-in dependencies in programmatic advertising protocols [172, 173, 146, 148, 149]. We contend that leveraging such fundamental information sharing patterns can help build accurate and robust classifiers for ad and tracker blocking. We introduce WebGraph, a classifier that explicitly captures these information sharing patterns as part of its cross-layer graph representation of the execution of a web page.

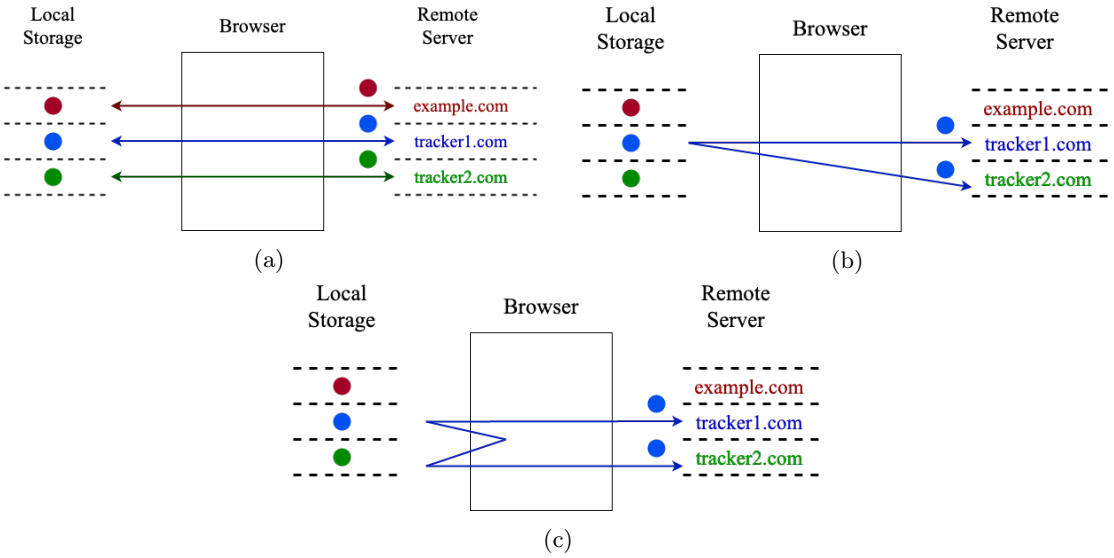


Figure 4.3: Origin isolation vs. sharing. Circles represent information about a user gathered by a particular domain (`example.com`, ●; `tracker1.com`, ●; and `tracker2.com`, ●). The box represents the browser which acts as channel between the local storage on the user's device and the remote server of each domain. (a) Illustrates origin isolation in the browser: every domain can only access information in their own storage. (b) and (c) illustrate two information sharing patterns that trackers use to circumvent origin isolation: (b) cookie syncing, where users' identifiers are sent to more than one domain; and (b) sharing identifiers using web APIs.

To illustrate the information sharing patterns that we want to capture in WebGraph, let us revisit how information sharing between different origins is mediated by the browser. We deliberately use a loose definition of origin. An origin can be, depending on the specific use case, a site, a domain, or an entity, among others. At a high-level, the web browser isolates different origins, based on various policies, so that their data is not leaked to each other. Figure 4.3(a) illustrates how the browser limits information sharing between different origins: `example.com`, `tracker1.com`, and `tracker2.com` each have access to their isolated local storage (e.g., cookies, IndexedDB) that may be used to store user

identifiers. The browser isolates information flows between the local storage and remote servers of different origins: `tracker1.com` and `tracker2.com` cannot generally access each others' cookies.

Trackers typically circumvent these limitations in the browser in two main ways. First, Figure 4.3(b) illustrates how a tracker may share its identifier with another tracker through cookie syncing. This can be implemented in several ways. For example, let's say `example.com` loads a JavaScript from Tracker 1 that first uses `document.cookie` to retrieve Tracker 1's identifier cookie from its cookie storage and then initiates a GET request to Tracker 2. The script includes Tracker 1's identifier cookie in the request URL as a query string parameter. Note that the request automatically includes Tracker 2's identifier cookie in the Cookie header. Therefore, when Tracker 2's remote server receives the request, it would be able to sync Tracker 1's identifier with its own identifier. As another example, let's say `example.com` first loads an invisible pixel from Tracker 1, which responds back with a 3XX redirect status code along with the URL in the Location header that points to Tracker 2 and includes Tracker 1's identifier cookie. Upon receiving the response, the browser issues a GET request to Tracker 2 and includes Tracker 1's identifier cookie in the request URL and Tracker 2's identifier cookie in the Cookie header. Again, Tracker 2's remote server is able to sync Tracker 1's identifier with its own identifier.

Second, Figure 4.3(c) illustrates how a tracker may share its identifier with another tracker through various web APIs in several ways. For example, let's say `example.com` loads scripts from Tracker 1 and Tracker 2 which then share their identifiers by reading/writing to the global variables of the `window` object. The script from Tracker 1 may assign its identifier to a new global variable `foo` that is then read by the script from Tracker 2. Therefore, Tracker 1 and Tracker 2's scripts would be able to sync identifiers with each other and also send them to their respective remote servers. As another example, let's say `example.com` loads iframes from Tracker 1 and Tracker 2 which then share their identifiers using `postMessage`. While these iframes have different origins, Tracker 1's iframe can use `window.parent` property to get a reference to the parent window and then use `window.frames` to get a reference to Tracker 2's iframe. Tracker 1's iframe can then use this reference to call `window.postMessage` and send its identifier to Tracker 2's iframe, which can use `window.addEventListener` to receive the identifier. Tracker 2's iframe can then send the shared identifier with its remote server to sync them.

Trackers use a wide variety of information sharing patterns, beyond the two aforementioned mechanisms. A sound and precise examination of all patterns warrants full-blown information flow tracking that adds significant implementation complexity and runtime overhead [174, 175, 176]. As we discuss next, WebGraph approximately captures these information sharing patterns by including additional nodes and edges in its graph rep-

resentation that correspond to elements and actions associated with these information sharing patterns. (See Section 4.6 for a discussion of WebGraph's completeness.) It then extracts new features on this enriched graph representation to train a classifier for detecting ads and trackers.

4.4.1 Design & Implementation

Graph Construction

WebGraph captures the flow of information among and across the HTML, network, JavaScript, and storage layers of the web stack. At the HTML layer, WebGraph captures creation and modification of all HTML elements that are initiated with scripts, e.g., `iframe`. At the JavaScript layer, WebGraph captures the scripts' interaction with other layer, e.g., initiation of a network request. At the network layer, WebGraph captures all outgoing network requests and their responses. At the storage layer, WebGraph captures read/write in cookies and local storage through scripts and network requests, and also value exchanges between network requests.

OpenWPM Instrumentation. We extend OpenWPM [117] to capture the execution and interaction of HTML, network, JavaScript, and storage layers. To capture HTML elements creation and modifications, we instrument `createElement` method and register a `MutationObserver` interface. To capture network requests, we parse OpenWPM's existing instrumentation, which uses a webRequest[2] listener, to capture all of the network requests, their responses, and redirects. To capture JavaScript interaction, we parse OpenWPM's existing instrumentation, which relies on JavaScript's stack trace to log JavaScript execution. To capture read/write to storage, we instrument `document.cookie` and localStorage methods and also intercept cookie read/write HTTP headers.

Graph Composition. Elements at each of the layers are represented with nodes and the interaction between these nodes is represented with edges. Specifically, each HTML element, network request, script, and stored value, is represented as a node. Edges to HTML nodes from script nodes represent the creation and modification of elements. Edges from HTML nodes to network nodes represent initiation of network requests to load content, such as scripts and images. Edges from script nodes to network nodes represent the initiation of `XMLHTTPRequest` which will be parsed by the script. Edges between script and storage nodes and network and storage nodes, represent the read/write of values in the storage. Edges between network nodes either represent redirects or the presence of the same stored values.[3]

[2]https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest
[3]We match stored values with their base64-encoded and MD5 and SHA-1 hashed values [177, 148].

Graph Composition Example. To illustrate WebGraph's graph representation, let us consider the example web page given by Code 4.1. The web page embeds a script from Tracker 1 and an iframe from Tracker 2. The tracking iframe from Tracker 2 reads its tracking cookies and sends them to Tracker 3 via an XHR. Both trackers trigger requests to share tracking identifiers. The HTTP requests and responses that result from loads in Code 4.1 are listed in Listing 4.1.

```
<html>
     <script src='tracker1.com/track.js'>
      ...
      var image =document.createElement('img');
      image.src = 'tracker2.com/sync';
      document.body.appendChild(image);
      ...
     </script>
     ...
     <iframe src='tracker2.com/track.html'>
      <script>
      ...
      idCookie = document.cookie;
      var newReq = new XMLHTTPRequest();
      newReq.open("GET", "tracker3.com?user_id=" + idCookie);
      ...
    </script>
  </iframe>
</html>
```

Code 4.1: Web page sending requests to several trackers.

Tracker 1's script embeds an image element from Tracker 2, which causes the browser to send an HTTP request (Request 1 in Listing 4.1) that includes Tracker 2's cookie. Tracker 2 responds to this request with a redirect to Tracker 1 that embeds the user identifier Tracker 2 received via the initial request's `Cookie` header (i.e., `user1`). The browser makes a subsequent request (Request 2 in Listing 4.1) to Tracker 1. Tracker 1 responds with a tracking pixel image and a `Set-Cookie` header to set its own tracking cookie with the value `userA`. On the backend, Tracker 1 knows that `userA` is known as `user1` by Tracker 2. Tracker 2's embedded iframe further shares its identifier cookie with Tracker 3. It does so by accessing its cookies locally via `document.cookie` and embedding them in an XHR to Tracker 3 (Request 3 in Listing 4.1).

Differences as compared to AdGraph. WebGraph keeps AdGraph's HTML and JavaScript layers as they are, but extends the network layer and includes a new storage layer in the graph representation. WebGraph also introduces information flow edges, which are absent in AdGraph, to entwine the extended network layer and the storage layer. The extension of network and the addition of storage layer allow WebGraph to explicitly capture information sharing patterns used in advertising and tracking.

```
1  ------------------------------------------------
2  Request 1
3  URL: tracker2.com/sync
4  Cookie: user1
5  Response 1
6  Status: 302
7  Location: tracker1.com?tracker2_id=user1
8  ------------------------------------------------
9  Request 2
10 URL: tracker1.com?tracker2_id=user1
11 Response 2
12 Status: 200
13 Set-Cookie: userA
14 Content: pixel.png
15 ------------------------------------------------
16 Request 3
17 URL: tracker3.com?user_id=user1
18 Response 3
19 Status: 200
```

Listing 4.1: HTTP requests and responses initiated from Code 4.1.

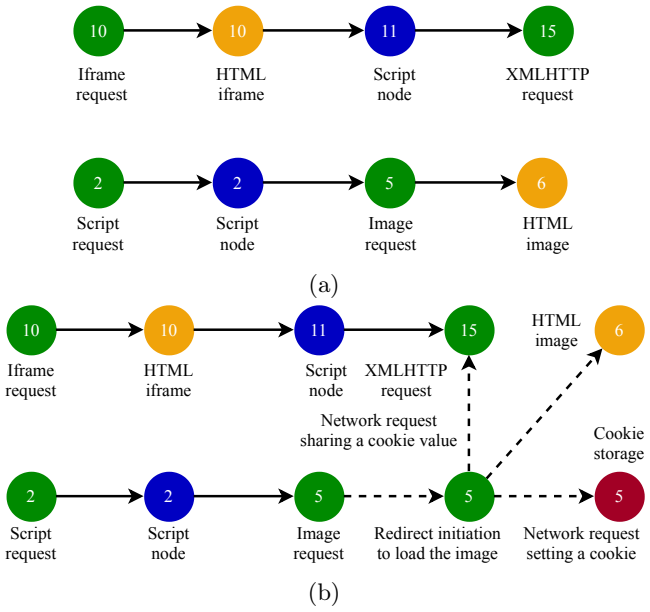


Figure 4.4: Graph representation of Code 4.1 in (a) AdGraph and (b) WebGraph. ● represents network nodes, ● represents script nodes, ● represents HTML nodes, and ● represents storage nodes. Node numbers correspond to the lines in Code 4.1. In (b), dotted (- - -) lines represent the additional edges that are captured by WebGraph and missed by AdGraph.

We illustrate the differences in Figure 4.4 which shows the graph representation of the web page in Code 4.1 and request and response sequences in Listing 4.1 for both AdGraph (Figure 4.4(a)) and WebGraph (Figure 4.4(b)). AdGraph's representation of the example web page consists in two disjoint graphs which capture the individual actions of the two trackers: The first row of nodes (from 10 to 15) captures Tracker 2's tracking behavior: from the iframe loading to the initiation of an XHR request. The second row of nodes (from 2 to 6) captures Tracker 1's tracking behavior: from the script loading to the initiation of a network request for loading an image. In this figure, it becomes clear that AdGraph *does not* capture the information sharing pattern between the nodes, because of its inability to capture the redirect (Request 2) made by the image request (network node 5) and the cookie set (storage node 5; visible only in WebGraph's graph) by the redirect request. WebGraph, on the contrary, not only captures the flows appearing in AdGraph, but also captures the redirects (dotted edge between the two network nodes labeled 5) and cookies set by requests (the second network node 5 to storage node 5). This representation further enables WebGraph to link requests that share common identifiers (node 5 to 15).

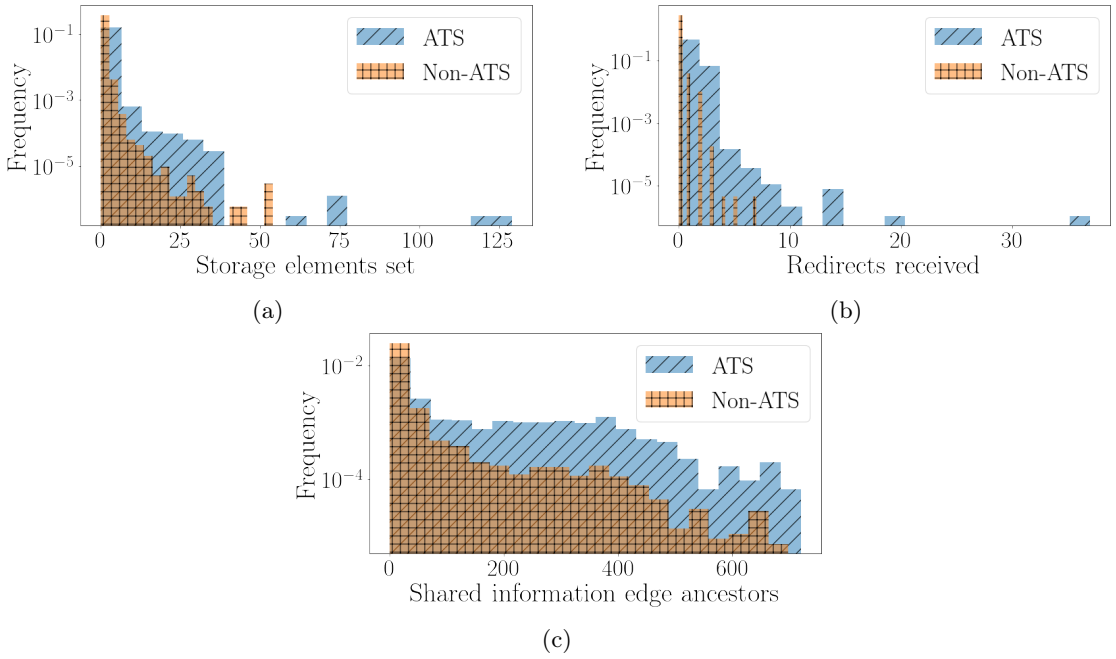


Figure 4.5: Histograms of three example flow features for ATS and Non-ATS resources (normalized, y-axis in log scale). (a) Number of storage elements set by a resource; (b) Number of network redirects received by a resource, and (c) Number of shared information ancestors of a resource. These features demonstrate different distributions for ATS and Non-ATS resources, and thus can help the classifier to distinguish between them.

Features

We take the AdGraph feature set and augment them with three categories of features. These additional features come from WebGraph's improved graph representation, i.e., extension of the network layer and a new storage layer. The features target storage, network, and information sharing behaviors that were absent in AdGraph. First, we extract features that measure the number of read/write cookie and localStorage accesses by a node. We obtain these features from the new storage layer. Second, we extract features that measure the number of requests and redirects to/from a node as well as the depth of a node in a redirect chain. These features come from our extension to the network layer. Third, we extract features that measure the number of different types of information sharing edges (e.g., nodes access the same storage node or share data of a storage node) to/from a node. We obtain these features using both the network and storage layers in WebGraph's graph representation. We also extract some standard graph features (e.g., in-degree, out-degree, eccentricity) for the information sharing edges. We jointly refer to these three newly added categories of features as *flow* features. Table C.1 in Appendix C.1 lists the full set of features in WebGraph.

To illustrate the potential of these features in distinguishing ATS and Non-ATS resources, let us consider three flow features belonging to each of the categories described above:the number of storage elements set by a resource (Figure 4.5(a)), the number of requests that were redirected to a resource (Figure 4.5(b)), and the number of information sharing edge ancestors (Figure 4.5(c)). As explained in Section 4.4, ATS resources store user identifiers in storage elements and use redirects and sharing of identifiers in URLs to perform actions such as cookie syncing. Therefore, we expect ATS resources to set a larger number of storage elements, be at the receiving end of redirects, and be involved in a larger number of shared information edges than Non-ATS resources. We plot in Figure 4.5 the distributions of these features in our dataset. We see that, indeed, the distributions are different for benign and ATS resources, with ATS presenting higher values on average for the three features under study. The differences in distributions is especially apparent for Figure 4.5(c), which shows the number of shared information edge ancestors. In our dataset, we observe 589,218 cases of ATS receiving a cookie value in a request URL, as compared to 89,564 cases for non-ATS. This sharing is detected as an information sharing edge, which in turn leads to ATS having larger values in shared information edge properties than Non-ATS. In the case of redirects (Figure 4.5(b)), the probability that a Non-ATS resource has more than 7 redirects tends to 0, which is not the case with ATS resources. The number of ATS resources with more than 7 redirects is very small in our dataset ($\approx 0.04\%$). Yet, it is a top-20 feature in our classifier, as observing more than 7 redirects directly identifies the resource as ATS. Storage element setting (Figure 4.5(a)) shows a similar behavior, with ATS resources sometimes having

more than 54 elements set, while Non-ATS resources never have so many.

While individual contributions of some of these flow features might be small, they provide a strong signal in distinguishing ATS when combined, as we show in the next section.

4.4.2 Evaluation

To evaluate WebGraph, we use the same dataset of 10K web pages and method as in Section 4.3.2. To understand the marginal benefit of WebGraph over AdGraph, we systematically compare the performance of different feature sets and graph representations. Table 4.2 summarizes the results.

We observe that AdGraph's performance drops by at least 10% when content features are removed. Recall from Section 4.3.2 that if content features are present alongside structural features, AdGraph is particularly susceptible to evasion: trackers have an 8.72% evasion success rate on their own, and a 96.62% success rate if they collude with the first party. Thus, there is a trade-off in AdGraph between effectiveness (with content) and robustness to evasion (without content).

Second, Table 4.2 shows that WebGraph's performance is better than AdGraph. When using all feature sets, WebGraph outperforms AdGraph by about 2-4%. If, for robustness, we remove content features, we observe a drop in accuracy limited to just 4-9% across all measures. We conclude that WebGraph's improved graph representation and new flow features can compensate for the loss of content features to a large extent.

Finally, Table 4.2 shows that WebGraph's improved graph representation by itself (i.e., even without the new flow features) contributes to about half of the improvement over AdGraph. WebGraph with only structural features achieves 2-4% improvement across all measures as compared to AdGraph also with only structural features. We conclude that, while WebGraph's new flow features help improve its accuracy, the improved graph representation is an important contributor to performance.

Graph	Feature Set	Accuracy	Precision	Recall
AdGraph	Structural + Content	92.33 ± 0.50	88.91 ± 1.14	92.14 ± 0.65
	Structural	80.22 ± 0.81	71.85 ± 1.53	82.44 ± 1.26
WebGraph	Structural+ Flow + Content	94.32 ± 0.27	92.24 ± 0.67	94.14 ± 0.30
	Structural + Flow	86.93 ± 0.64	80.57 ± 1.12	90.01 ± 0.50
	Structural	82.62 ± 0.47	75.67 ± 0.75	85.09 ± 1.41

Table 4.2: Evaluation of WebGraph and AdGraph with different feature set variations.

To provide insights into the relative importance of flow and structural features, we

list top five most important features in terms of information gain in Table 4.3. The two most important features are flow features. As discussed in Section 4.4.1, the top feature distribution (Figure 4.5(c)) is very different for ATS and non-ATS, so it's not surprising that this feature contributes to the classification. Storage setting (Figure 4.5(a)) and received redirects (Figure 4.5(b)) contribute a smaller, but still useful, portion towards identification; they have information gains of 1.9% (± 0.37) and 2.5% (± 0.47) respectively (21st and 17th most important features). Structure features, enhanced by WebGraph's improved graph representation, also contribute towards the performance. We further analyze which features contribute most to each prediction of WebGraph using `treeinterpreter` [178]. For ≈ 32% of predicted ATS in the dataset, the flow features were the top contributors, indicating that they provide an important signal for the presence of trackers. In contrast, for ≈ 47% of predicted Non-ATS in the dataset, structure features were the top contributors. These results confirm our earlier intuition that capturing information sharing behaviors that are unique to advertising and tracking carries significant predictive power.

Feature	**Category**	**Information gain (%)**
Shared information ancestors	Flow	6.48 ± 0.69
Number of requests sent by node	Flow	5.9 ± 0.69
Number of nodes in graph	Structure	5.46 ± 0.35
Average degree connectivity of node	Structure	5.18 ± 0.16
Number of edges in graph	Structure	4.19 ± 0.34

Table 4.3: Top-5 most important features for WebGraph's classification, their category, and information gain (averaged over 10 folds).

4.4.3 Efficiency

We envision WebGraph to be used for filter list curation and maintenance in an offline setting. WebGraph relies on large scale web crawls and notoriously expensive graph traversals for feature extraction. We now measure WebGraph's offline overhead to demonstrate its adequacy as a tool to periodically update filter lists.

Crawl time. Our implementation of WebGraph has an upper bound of 60 seconds, enforced with a timeout, to crawl a website. In the average case, crawls take only ~26.46 seconds per-page. Crawls can be parallelized over several instances to reduce the crawl time. For example, it took us around 10.5 hours to crawl 10K websites, parallelized over 7 instances. Without parallelization and if all websites would reach the timeout, the crawls would take ~166 hours.

Processing websites. On average, WebGraph takes 0.72 seconds to build the graph, 15 seconds to extract features, and 0.25 seconds to train and test each website. For our crawl of 10K websites, it took us a total of ~44 hours to create their graphs and extract features on a single instance. This time can be significantly reduced with parallelization.

Update frequency. These estimates suggest that for 10K websites containing ~1.1 million requests WebGraph will require, at most, ~166 (data crawling) and ~44 (data processing) hours with a single instance. However, when averaged over 7 instances, the computation time significantly reduces to 16.83 hours (10.5 for crawling and 6.33 for processing). We anticipate the computation time for periodic updates to reduce significantly because many websites have low update frequency. Monitoring the update frequency of websites will allow us to only crawl when changes are expected in websites. In cases where we determine that the website did not change since the last crawl, we will not recompute their classifications. With this update frequency, WebGraph will be able to update filter lists on a daily basis, and certainly operate within the current expiry period, i.e., mandated update frequency, of popular filter lists, e.g., 4 days for Easylist [131]. Frequent updates with WebGraph can help remove outdated rules and as well as add new rules to block newly discovered ads and trackers.

To evade detection by the updated rules generated by WebGraph, adversaries could change their page content (e.g., rotate domains) [147]. In order to successfully evade WebGraph, however, an adversary would need to change their page content at a rate faster than the rate of WebGraph's updates (i.e, at least once a day). Such frequent changes, however, require continuous coordination and cooperation between the trackers and publishers hosting their content and are complicated to implement in practice. If an adversary with sufficient resources and capabilities to perform frequent page content changes, WebGraph would need to operate in an online manner to be robust by directly classifying page resources at run-time. This would require few changes in WebGraph's operation. Specifically, instead of extracting features from a complete graph representation at the end of a page load, in an online setting WebGraph would need to extract features from partial graph representations build as the page is being loaded. Relying on partial graph representation, will make WebGraph more performant, but it may may degrade WebGraph's accuracy. We leave to future work to explore the trade-offs involved in an online implementation of WebGraph.

4.5 WebGraph Robustness

In this section, we evaluate WebGraph's robustness against content mutation attacks (described in Section 4.3) and structure mutation attacks.

4.5.1 Content mutation attacks

To evaluate WebGraph against content mutations, we strengthen the threat model described in Section 4.3 to enable the adversary to also masquerade their resources as first party, i.e., through first-party subdomains. Overall, our attacks involve random mutations to domain names, subdomains, and the query string in URLs (Section 4.3.2).

By relying on content mutations, the adversary is able to switch 96.62% of their `ATS` resources to `Non-ATS` against AdGraph. Against WebGraph, the adversary's success rate plummets to just 8.34 ± 0.66% (over 10 folds). For example, `mylivesignature.com`, a tracking domain, was able to switch all of its 560 `ATS` resources to `Non-ATS` against AdGraph, but none against WebGraph.

Note that, even though WebGraph does not use content features, the evasion success rate against WebGraph does not drop to zero. This is because some of the WebGraph's features implicitly rely on URL properties. For example, shared information edges, that consider sharing of cookie values via query strings in the URL, are affected by URLs manipulations.

4.5.2 Structure mutation attacks

Next, we evaluate WebGraph's robustness against structure mutations. We assume that the adversarial third-party has unrestricted black box access to the WebGraph's classifier, i.e., the adversary can make unlimited queries and observe WebGraph's classification output. This access enables the adversary to validate the effect of their structure mutations.

Attack details. We assume that the adversary can mutate the structure of a web page through resource addition, re-routing, and obfuscation. Moreover, we assume that the adversary also performs content mutations, to maximize its chance of success Resource addition entails addition of new resources, such as images and scripts. Resource re-routing entails re-organization of existing redirect chains, i.e., dispersing a redirect chain in a sequence of `XMLHttpRequest's` through one or multiple scripts. Resource obfuscation entails obfuscation of cookie or query string parameter values of existing resources, i.e., encoding or encrypting cookie or query string parameter values in a format that is not detected by WebGraph's implementation, before sharing them in network requests. To remain stealthy, we assume that the adversary does not delete functional content from the web page that could damage usability.

It is important to note that even simple mutations, such as adding a single element to the web page, can significantly change graph properties and impact several features. For

example, the addition of a child node causes a cascading effect. It increases the number of descendants of all the parent nodes in the branch, all the way up to the root node, and also impacts their centrality. Thus, the result of such simple mutations can become unpredictable and hard to control by the adversary: It can cause unintended classification changes for nodes under and outside the control of the adversary. Complex mutations, such as adding a combination of nodes at once, further complicate having control on the number of unintended classification changes. In our evaluation, we only consider atomic mutations, i.e., addition, re-routing, or obfuscation of individual resources.

Mutation algorithm. We capture the adversary's unrestricted black box access to classifier by implementing a greedy random algorithm to find suitable mutations. This kind of algorithm is extensively used in the literature due to its simplicity and practicality [179, 180, 181]. The algorithm (formally described in Appendix C.4) iteratively mutates WebGraph's graph representation. At each step, it adds, re-routes, or obfuscates the resource that provides the best trade-off between *desired* (`ATS` to `Non-ATS`) and *undesired* (`NON-ATS` to `ATS`) classification switches. Resource addition is simulated by adding nodes to a randomly selected leaf nodes in the graph. Resource re-routing is simulated by adding each request, in a redirect chain, as an individual node to one or more randomly selected scripts. Resource obfuscation is simulated by replacing stored values in URLs with an encoding that is not detected by WebGraph.

4.5.3 Empirical evaluation

Experimental Setup. To evaluate WebGraph's robustness, we must rebuild the graph and recompute the features after each mutation. To keep the evaluation time reasonable, we sample 100 web pages from our dataset, and we limit the graph growth to 20%. To ensure that this sample is representative of our dataset, we divide graphs into 5 bins according to their size and sample 20 web pages from each bin. To avoid exceptionally long evaluation times, we only consider web pages that have 250 or fewer nodes (80% of the dataset; see Appendix C.2 for the full distribution).[4] For each web page, we designate the adversary as the third party with the highest number of resources classified as `ATS`. It is noteworthy that the adversary with the highest number of `ATS` resources has an opportunity to do maximum damage.

In this dataset, the median evaluation time per web page was 29.08 minutes, with 39% of the pages taking more than an hour to run. Even though this is a simulation, the computational cost is directly proportional to the operational cost for the adversary. The adversary must consume additional CPU cycles and memory and in the case of

[4]The resulting reduced dataset has similar mean, stand deviation, and median for the features as the full dataset.

node addition, send additional network requests, thereby increasing the cost of their attack.

Success metrics. To measure adversary's success, we define the following terms:

$\texttt{ATS}_{\texttt{Web}}$: Number of nodes classified as `ATS`.

$\texttt{ATS}_{\texttt{Adv}}$: Number of adversary nodes classified as `ATS`.

$\texttt{Non-ATS}_{\texttt{Web}}$: Number of nodes classified as `Non-ATS`.

$\texttt{Non-ATS}_{\texttt{Adv}}$: Number of adversary nodes classified as `Non-ATS`.

`desired`: Number of nodes switching from $\texttt{ATS}_{\texttt{Adv}}$ to $\texttt{Non-ATS}_{\texttt{Adv}}$.

`undesired`: Number of nodes switching from $\texttt{Non-ATS}_{\texttt{Web}}$ to $\texttt{ATS}_{\texttt{Adv}}$.

`neutral`: Number of nodes switching from `ATS` to `Non-ATS` for non-adversary nodes.

Success rate: Desired changes from the adversary's point of view. It is calculated as $\texttt{desired}/\texttt{ATS}_{\texttt{Adv}}$.

Collateral damage: Undesired changes from the adversary's point of view. It is calculated as $\texttt{undesired}/(\texttt{Non-ATS}_{\texttt{Adv}} + \texttt{Non-ATS}_{\texttt{Web}})$.

Other changes: Non-consequential changes from the adversary's point of view. It is calculated as $\texttt{neutral}/\texttt{ATS}_{\texttt{Web}}$.

We illustrate the node switches, with the mutation algorithm, for an example graph in Appendix C.5.

Adversary's success

We assume that the adversary neither colludes with other third parties nor with the first party and can only perform mutations on the nodes and edges it controls. We conduct the attack on 100 web pages. We note that increasing the number of graph mutations increases the adversary's mean success rate from 38.6 ± 33.01 (median: 33.33) at 5% graph growth to 52.48 ± 33.4 (median: 50.00) at 20% graph growth. The classification switches lead to a decrease in the overall classification accuracy by 1.5%, recall by 8.85%, and precision by 2.29%.

However, the adversary's success comes at a cost of collateral damage. The average collateral damage rises from 2.17 ± 11.19 to 3.88 ± 13.55 (median: 0). In Figure 4.6a, we

illustrate the trade-off between success rate and collateral damage at 20% graph growth. The x-axis represents success rate, the y-axis represents collateral damage, and circles represent a trade-off between the two. The circles' color represents $\texttt{ATS}_{\texttt{Adv}}$ or the number of classifications the adversary has to switch for the particular web page. The lighter the color, the more switches are required, i.e, the cost of success increases. For the web pages in this dataset, the adversary has, on average, $\texttt{ATS}_{\texttt{Adv}} = 5.98 \pm 5.39$ nodes classified as $\texttt{ATS}$. For certain pages, the $\texttt{ATS}_{\texttt{Adv}}$ can be as high as 26.

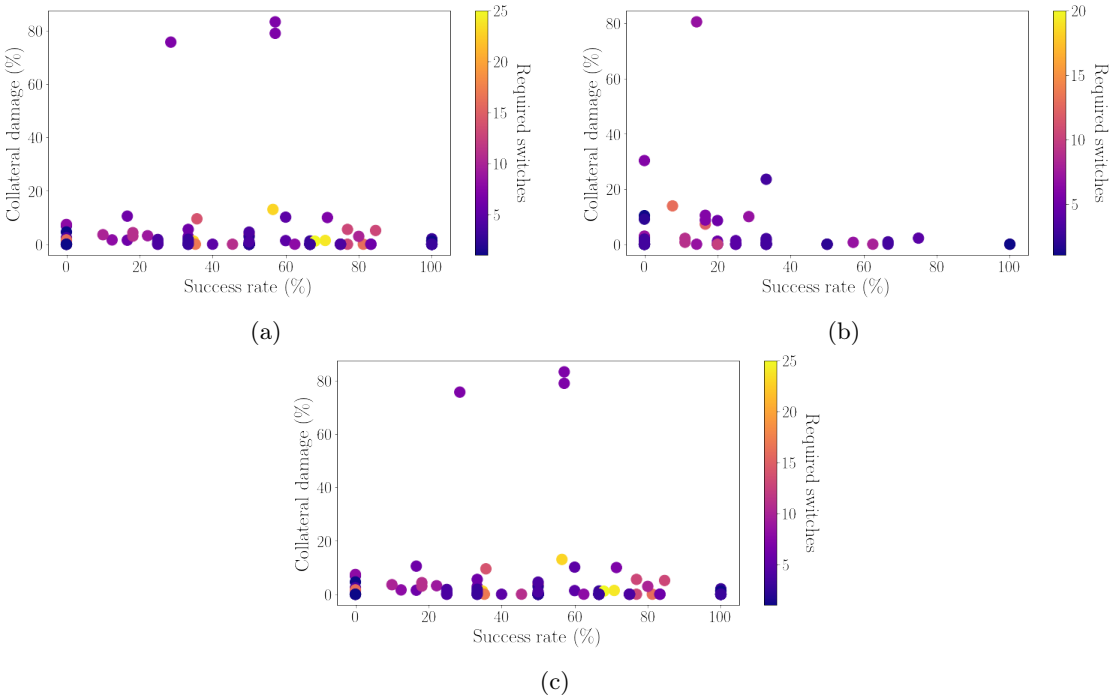


Figure 4.6: Adversary's success rate vs. collateral damage for each web page in the test data at 20% graph growth. (a) represents all mutations, (b) represents only structure mutations, and (c) represents only resource re-routing and obfuscation mutations. Colored circles represent the number of required switches.

Ideally, the adversary wants to be at the bottom right of the graph, where it achieves 100% success rate with zero collateral damage. The adversary is able to reach its ideal target on only 13 web pages, which only required four switches. The adversary is able to achieve 50% or more success on 61 of the tested web pages. Together, they amount to 240 nodes switched, with 45 of these pages having non-zero collateral damage. On the other hand, we have 9 web pages that had a higher collateral damage than success rate: a net negative effect of the mutation. Out of these, 6 web pages had 0% success rate with non-zero collateral damage, and 3 web pages had a large collateral damage > 75% (with one web page hitting 83%).

Overall, even in the case of an unrealistic adversary that has the capabilities to manipulate structure features at will, and also the operational power to do so for a large number of iterations, there is no guarantee of perfect success.

Breakage. If undesired changes affect benign resources that are essential to the correct functioning of the web page, even a small collateral damage can break the page. This may have large impact on trackers. If users leave the broken web pages, the adversary cannot track them or show them ads.

We define website breakage as degradation in usability of the website. We say there is `major` breakage if the user is unable to complete the primary functionality of the web pages (e.g. login, search or page navigation). If the user is unable to complete a secondary functionality of the web pages (e.g. comment or review), we say there is `minor` breakage. Otherwise, we consider that the web page does not have any breakage.

We quantify breakage on all of the 21 web pages where the adversary experiences undesired classification switches.[5] We open these web pages side by side on stock Firefox and a Firefox configured with an extension that blocks the URLs that switched classification, and we compare them side by side to identify any visual signs of breakage.

We ask two reviewers to perform the analysis. Our reviewers attain an agreement of 90.46% in their evaluation. They find that the undesired classification switches cause `major` breakage on 3 and `minor` breakage on 2 web pages. This breakage mostly happens when the first-party resources are switched from `Non-ATS` → `ATS`. Our approach and results are in line with other works that evaluate breakage [142, 182], though we cannot test whether this breakage is representative of what users experience in the wild.

Careless adversary. If the adversary is not concerned with changes to any non-adversarial nodes, their collateral damage decreases. The adversary still does not want their own content to be blocked, so it will optimize against their own nodes switching to `ATS`. This change in strategy updates the collateral damage calculation to: $\texttt{undesired}/\texttt{Non-ATS}_{\texttt{Adv}}$.

With our modified definition, there can be no collateral damage for web pages on which all of the adversary nodes are classified as `ATS`; we note 55 such web pages. For the remaining 45 web pages, only 8 web pages have collateral damage, as compared to 27 web pages that had collateral damage as per our original definition. Out of these 8 web pages, 4 had a higher collateral damage than success rate (net negative effect), and 6

[5]The adversary experiences undesired classification switches on 45 web pages. However, 24 web pages no longer serve the switched `ATS` resources.

web pages have a large collateral damage > 20% (with 2 web pages hitting 100%). Thus, even when an adversary is not concerned about collateral damage to other parties they are not significantly more successful in subverting WebGraph. Surprisingly, the success rate mean growth does not change much from the previous scenario. The unpredictable effect of the mutations on the graph features (see Appendix C.5 for an example) makes it difficult to pinpoint what causes this lack of change in the adversary's success rate.

Collusion with the first party. So far, we have assumed that the adversary is a single third party that does not collude with other third parties or the first party. If we assume the adversary colludes with both, the adversary can add child nodes to *any* node in the graph. This is a much stronger adversary than in Section 4.5.3, where in each iteration the adversary can only test a random subset of the options. Realistically, such a powerful collusion would be difficult to implement, as it would require coordination and cooperation among multiple parties to ensure that the mutation is feasible.

We repeat our experiment, but we now allow the adversary to consider all possible mutation options on any node, and pick the best one in each iteration. These experiments take longer to run (see Appendix C.3), so we only analyze 100 web pages whose graphs have at most 50 nodes. We see that collusion enables the adversary to have a slightly higher success rate (63 pages with success rate > 50% as compared to 60 for the non-colluding adversary) and lower collateral damage (9 pages with damage >0% compared to 18 pages for a non-colluding adversary). These results are described in detail in Appendix C.6).

Impact of mutation choice

Next, we study the adversary's preference in selecting the most useful mutations. The adversary picks resource addition 81.70%, resource re-routing 17.26%, and resource obfuscation 0.04% of the time. Resource obfuscation is rarely chosen by the adversary because the graph already has content manipulations applied, and these manipulations have already severed many of the edges that would be severed by resource obfuscation. To separate out the impact of different mutations, we conduct two additional experiments: (1) where the adversary can only perform resource addition, and (2) where the adversary can only perform resource re-routing and obfuscation.

We exclude 33 of the web pages for experiment 2 because these web pages do not have re-route-able or obfuscate-able resources. For the remaining 67 pages, we see that the re-routing/obfuscation mutations (Figure 4.6(b)) are more effective than addition mutations (Figure 4.6(c)). Re-routing/obfuscation not only yields higher success rates for the adversary, but also results in lower collateral damage. This is unsurprising because these

mutations target information sharing patterns which are distinctive of trackers; changing these patterns removes an important signal for the classifier (see Table 4.3).

However, in practice, resource re-routing and obfuscation would entail high costs for the adversary since they involve the manipulation of identifier sharing patterns. Specifically, the adversary would have to coordinate with other parties on changes to these patterns, and redesign how they perform tracking in order to perform these mutations. The success of these mutations also depends on the degree to which flows are captured by the instrumentation used to create the graph. WebGraph's instrumentation approximates information flows and will not capture all attempts by an adversary to use re-routing and obfuscation. We argue that this is not a fundamental flaw in WebGraph's architecture but a limitation in our implementation that approximates information flow (Section 4.4). A fully fledged instrumentation would make these manipulations much more difficult to deploy. See Section 4.6 for an extended discussion. Resource addition has fewer costs for the adversary since it does not involve coordination with additional parties. This manipulation is not affected by the type of implementation because it is not related to the flow of identifiers.

Comparison with AdGraph

We also evaluate whether WebGraph, in addition to having superior classification performance, offers robustness benefits over AdGraph. We only use AdGraph's structural features, as we already demonstrated that content features are not robust. Because AdGraph does not have features based on flow information, we only perform resource addition. We find that the adversary has greater success against AdGraph than WebGraph, but also suffers from more collateral damage (Figure C.4 in Appendix C.6). This is because the structural effects of node additions are hard to control, as explained in Section 4.5.2. Since the former is beneficial to the adversary but the latter is not, it is not clear-cut as to whether one system provides more robustness than the other.

In summary, it is not trivial for an adversary to produce the desired classification switches for their advertising and tracking resources without producing any undesired changes. Yet, an adversary willing to accept the collateral damage, and with the resources to grow the graph beyond the 20% we evaluated, can increase the success rate. Even this success, however, would increase the adversary's operational and computational costs.

4.6 Limitations

In this section, we discuss limitations of WebGraph's design, implementation, and evaluation.

Completeness. For efficiency reasons, WebGraph focuses on a limited subset of the browser's API surface, such as HTTP cookie headers, `document.cookie`, and `window.localStorage`. WebGraph's implementation is also geared towards capturing client-side information that is pertinent to stateful tracking. However, techniques used by ATS need not to be limited to these APIs or to stateful tracking. Some ATS have started to use stateless tracking techniques, such as browser fingerprinting, which use APIs that are not currently covered by our instrumentation [183, 184, 185]. To account for these techniques, WebGraph's instrumentation must be extended to include the corresponding APIs.

WebGraph's manually designed graph representation and feature set capture the most well-known information sharing patterns. The limits of these approach are shown in Section 4.5.3, where we show that an adversary capable of hiding or obfuscating traditional sharing flows has a better chance to bypass WebGraph than doing structure modifications. This limitation is, however, linked to our implementation choices. To increase WebGraph's coverage of sharing behaviors, if suffices with increase the instrumentation to cover more information flows. Ideally, we would instrument full-blown information flow tracking. Such expansion would incur prohibitive run-time overheads (up to 100X-1000X [174]) and its complexity makes it hard to integrate in the browser [175, 176, 186, 187]. Nevertheless, the design of WebGraph permits that the instrumentation to be upgraded gradually, as ATS evolve in response to our evasion protection techniques, increasing the cost of evasion without fundamentally changing the detection approach.

Robustness analysis. Inspired by previous work on graph-based detection evasion [179, 180, 181], we use a greedy algorithm to attack WebGraph. This algorithm only considers the best mutation in each iteration. Thus, it is not guaranteed to find the optimal mutation sequence that would lead to the best adversary performance. We note however that even exhaustive search does not lead to perfect success (see our results on small websites). We expect adversaries to try alternative algorithms to improve their success rates. However, any alternative that is close to exhaustive search will become prohibitively expensive for the adversary when the web page graph is large.

Another option for the adversary would be to perform more sophisticated graph mutations instead of the simple node additions that we perform. An adversary could tailor their mutations to the page's graph structure by studying how their node changes affect the graph properties of the web page. However, this requires that the rest of the graph (i.e., the portions outside of the adversary's control) remaining unchanged. Realistically, it would be difficult for an adversary to coordinate with other parties to generate these changes.

Finally, we note that the dynamism of modern websites [188] complicates the process for the adversary. Web pages change often, sometimes on every load. Even if the adversary manages to find an appropriate set of mutations, those mutations may be invalid the next time the page is reloaded.

4.7 Conclusion

In this chapter, we showed that state-of-the-art ad and tracker blocking approaches are susceptible to evasion due to their reliance on easy-to-manipulate content features. We then showed that information sharing patterns in online advertising and tracking can be leveraged for their robust blocking. We proposed WebGraph, that builds a cross-layer graph representation to capture such information flows and trains a machine learning classifier for accurate and robust blocking. Our results showed that it is non-trivial to evade WebGraph's classifier without causing unavoidable collateral damage.

While it is not foolproof, WebGraph raises the bar for advertisers and trackers attempting to evade detection. We foresee that advertising and tracking services would need to significantly re-architecture their information sharing patterns to achieve long-lasting evasion against WebGraph. We note, however, that introducing new information flows may be quite complicated, as they may require collaboration among the first-party and numerous third-parties on a typical web page.

5 CookieGraph: Measuring and Countering First-Party Tracking Cookies

This chapter is based on the article:

"CookieGraph: Measuring and Countering First-Party Tracking Cookies.", S. Munir, S. Siby, U. Iqbal, S. Englehardt, C. Troncoso, Z. Shafiq, *under submission*, 2022.

5.1 Introduction

Browser vendors and trackers are engaged in an arms race. As soon as browser vendors deploy privacy protections, *e.g.*, third-party cookie blocking [189, 190], trackers quickly adapt to evade them, *e.g.*, CNAME cloaking [145], bounce tracking [191] *etc.* In response, browser vendors have developed targeted countermeasures against such evasions [192, 193].

To gain advantage over browser vendors, trackers have started exploiting browser features that are typically used for functional purposes, and thus cannot be trivially blocked. The abuse of JavaScript APIs to build identifiers, *i.e.*, browser fingerprinting [194], and the abuse of first-party context to store tracking cookies [195] stand out as two prominent techniques. Browser vendors have largely struggled against these tracking techniques because preventing them requires compromising functionality [196, 189].

While these new tracking techniques are difficult to counter, they do not offer the same flexibility as third-party cookies for cross-site tracking. Browser fingerprints enable cross-site tracking but are not stable over time [197]. On the other hand, first-party cookies are stable but not linkable across different sites. When combined, however, browser fingerprints and first-party cookies complement each others' shortcomings and enable reliable cross-site tracking. Specifically, trackers are able to leverage non-deterministic

fingerprints in the first-party context to set deterministic first-party tracking cookies [198].

Prior literature has shown how first-party cookies set by third party scripts are exfiltrated to tracking endpoints [195, 199, 148] and how trackers use browser fingerprinting to respawn first-party cookies [198]. While prior work has demonstrated that first-party cookies are indeed abused by advertisers and trackers, no countermeasure has been proposed to specifically block first-party tracking cookies.

In this chapter, we investigate how first-party cookies are abused for cross-site tracking and use our findings to develop a machine learning based countermeasure, CookieGraph, to block first-party tracking cookies.

To this end, we first perform a differential measurement study where we compare the first- and third-party cookie usage from two crawls of 10K websites when third-party cookies are enabled and blocked. We show that third-party-cookie blocking does not significantly impact sharing of identifiers to tracking endpoints because trackers use (or reactively shift to) first-party cookies when third-party cookies are blocked. Specifically, we find entities such as Criteo, Lotame, and ID5 that show an *increased* presence and reactively move to first-party cookies when third-party cookies are blocked. Our further analysis reveals that they store identifiers in first-party cookies, based on probabilistic and deterministic attributes, which can be then used for cross-site tracking.

Unlike third-party cookies, blocking all first-party cookies is not practical because it would lead to major breakage of legitimate website functionality. Privacy-enhancing content blocking tools, which use crowdsourced filter lists or machine learning [142, 143, 200], could be an alternative since blocking requests would also block all cookies set by the requests (or the requested scripts). However, as we also find in our evaluation, blocking requests would also lead to breakage since it is likely that many of the blocked cookies are needed for legitimate website functionality. Researchers have recently started to develop approaches to detect and block tracking cookies (both first and third-party) [201, 202]. However, these approaches rely on content-based features such as cookie names and values, which can lead to high number of false positives (and consequently higher major website breakage) while also being susceptible to evasion [200].

Keeping these limitations in mind, we design and implement CookieGraph, a machine learning approach to detect first-party tracking cookies. Instead of using content-based features, CookieGraph captures fundamental tracking behaviors exhibited by first-party cookies that we discover in our differential measurement study. CookieGraph is able to detect first-party tracking cookies with 91.06% accuracy, outperforming the state-of-the-art CookieBlock [202] approach by 10.28%. We also show that CookieGraph does not

cause any major website breakage, where CookieBlock causes major breakage on 8% of the websites with SSO logins. Moreover, CookieGraph is robust to evasion through cookie name manipulation while CookieBlock's accuracy degrades by 15.68%.

Our deployment of CookieGraph on 10K websites shows that first-party tracking cookies are used on 93.43% of the websites. While first-party tracking cookies are set by third-party scripts served from a total of 1,588 unique domains, we show that the most prevalent first-party tracking cookies are set by major advertising entities such as Google, as well as many specialized entities such as Criteo. We also show that 41.45% of all first-party tracking cookies are set by scripts served by domains involved in fingerprinting.

In summary, our key contributions are as follows:

1. We conduct a **large-scale differential measurement study** to understand the effectiveness of third-party cookie blocking and whether first-party cookies are used in lieu of third-party cookies.

2. We design and implement CookieGraph, **a machine learning based countermeasure** to detect and block first-party tracking cookies. CookieGraph captures fundamental tracking behaviors of first-party cookies that we discovered in our measurement study, and outperforms the state-of-the-art in terms of accuracy, robustness, and breakage minimization.

3. We **deploy CookieGraph on 10K websites** sampled from the Alexa's top-100K list to measure the prevalence of first-party tracking cookies. We detect a total of 1,588 distinct domains that set first-party tracking cookies, including major advertising entities such as Google, and show that 45 (2.83%) of these domains are known fingerprinters which set 41.45% of all first-party tracking cookies.

5.2 Background & Related Work

5.2.1 Adoption of third-party cookies for tracking

While cookies were originally designed to recognize returning users, e.g., to maintain virtual shopping carts [203], they were quickly adopted by third-parties to track users across websites, e.g., to serve targeted ads [204]. Early standardization efforts mostly focused on limiting unintended cookie sharing across domains [205] and, despite well-known privacy concerns [206], largely ignored the intentional misuse of cookies by third-parties for cross-site tracking. Over the years, the use of third-party cookies for cross-site tracking has become increasingly prevalent [207, 208, 195, 209]. Prior research has found that the vast majority of third-party cookies are set by advertising and tracking services [209] and

that the third-party cookies outnumber first-party cookies by a factor of two [208] and up to four when they contain identifiers [195].

5.2.2 Countermeasures against third-party cookies

Safari

Since its inception in 2003, Safari has blocked third-party cookies from domains that have not been visited by the user as full-fledged websites [189]. To strengthen its cookie blocking, Safari introduced Intelligent Tracking Prevention (ITP) in 2017. ITP used machine learning to automatically detect third-party trackers and revoked storage access from classified domains if users did not interact with them on a daily basis (i.e., a 24 hour period) [210]. Since 2017, ITP went through several iterations, i.e., ITP 1.1 [211], ITP 2.0 [212], ITP 2.1 [213], ITP 2.2 [214] and ITP 2.3 [215], eventually leading to full third-party cookie blocking [216].

Firefox

Firefox experimented with third-party cookie blocking in 2013 [217, 218], but did not ship default-on third-party cookie blocking until the release of Enhanced Tracking Protection (ETP) in 2018 [219]. ETP blocks third-party cookies based on a blocklist of trackers provided by Disconnect [220]. As of 2022, Firefox has launched Total Cookie Protection (TPC) which partitions all third-party cookie access [221]. Partitioning ensures that cookies set by a third-party on one site are distinct from those set by the same third-party on other websites, eliminating the third-party's ability to track users across those websites.

Internet Explorer and Microsoft Edge

Amongst the mainstream browsers that have deployed countermeasures against third-party cookies, Internet Explorer (IE) and Microsoft Edge have the most permissive protections. IE blocked third-party cookies from domains that did not specify their cookie usage policy with P3P response header [222]. However, website owners often misrepresented their cookie usage polices, which rendered P3P ineffective [223]. Since 2019, Microsoft Edge blocks access to cookies and storage in a third-party context from some trackers, based on Disconnect's tracking protection list [129, 224, 220].

Chrome

Google Chrome is the only mainstream browser that does not restrict third-party cookies in any way in its default mode. In 2020, Google announced plans to phase out third-party

cookies in Chrome by 2022 [225]. However, the plan has been postponed several times and the latest timeline suggests the phasing out of cookies by late 2024 [226]. Google has also announced plans to implement privacy-preserving versions of advertising use cases that currently depend on third-party cookies—including behavioral ad targeting and ad attribution/measurement [226].

5.2.3 Adoption of first-party cookies for tracking

While third-party cookies are widely considered as the main mechanism for cross-site tracking, trackers have also relied on first-party cookies for tracking. As early as 2012, Roesner *et al.* [207], noted that third-party tracking scripts, embedded on the main webpage (i.e., in first-party context), set first-party cookies. More recently, in 2020 Fouad *et al.* [148] found that trackers sync first-party cookies to several third-parties on as many as 67.96% of the websites. In 2021, Chen *et al.* [199] found that more than 90% of the websites contain at least one first-party cookie that is set by a third-party script. Similar to Fouad *et al.*, they also found that at least one first-party cookie is exfiltrated to a third-party domain on more than half of the tested websites, raising concerns that these cookies might be used for tracking. These concerns were also echoed by Sanchez *et al.* [195], who uncovered several instances where different third-parties interacted with the same first-party cookies. They conclude, through a large scale measurement study of top websites and a number of case studies, that even after blocking third-party cookies, users are still at risk of tracking through first-party cookies.

While prior studies have identified the use of first-party cookies by trackers, they were not solely focused on studying first-party tracking cookies. In fact, their measurement infrastructure was not designed to capture tracking through first-party cookies. For example, they did not configure their browsers to block third-party cookies, which might not instigate trackers to use first-party cookies for tracking.

Techniques used to set first-party cookies. It is non-trivial to generate first-party identifiers that are accessible across websites. Prior research has found that trackers often leverage browser fingerprinting to generate first-party tracking cookies [198]. Browser fingerprinting provides unique identifiers that are accessible across websites but drift over time [227]. However, identifiers generated through browser fingerprinting can be stored in cookies that persist even after fingerprints change. In addition to browser fingerprinting, several advertising and tracking services, such as Google Ad Manager [228, 229] and ID5 [230], specify in their documentation that they also use publisher provided identifiers (PPIDs), such as email addresses, to set first-party cookies.

CNAME cloaking also allows advertisers or trackers to use first-party cookies. In this work,

we do not focus on CNAME cloaking because first-party cookie leaks due to CNAME cloaking is already extensively studied by prior work [231, 145].

5.2.4 Countermeasures against first-party cookies

Deployed countermeasures

Safari is the only mainstream browser that has deployed protections against first-party tracking cookies. Safari's ITP expires first-party cookies and storage set by scripts in 7 days if users do not interact with the website [189]. For first-party cookies, this limit is lowered to 24 hours if ITP detects link decoration being used for tracking [189]. However, first-party cookie tracking does not require link decoration to be effective. In cases where link decoration isn't used, trackers can still track users within the 7-day window and beyond if users interact with the website within the 7-day window.

Countermeasures proposed by prior research

Recently, researchers have proposed machine learning based approaches to detect first-party and third-party tracking cookies. Hu *et al.* [201] developed a machine learning based approach that uses sub-strings in cookie names (e.g., track, GDPR) as features to detect first-party and third-party tracking cookies. Bollinger *et al.* [202] also developed a machine learning approach, CookieBlock, that uses several cookie attributes such as the domain name of the setter, cookie name, path, value, expiration, etc, as features to detect first-party and third-party tracking cookies. However, relying on hard-coded content features make these approaches susceptible to adversarial evasions (as we show later in Section 5.4.4). Moreover, these approaches mainly rely on self-disclosed cookie labels as ground truth which are known to be unreliable [232].

Request blocking approaches

Request blocking through browser extensions, such as Adblock Plus [150], and machine learning based tracker detection approaches proposed by prior research, e.g., [200], can potentially block first-party tracking cookies. However, request blocking is inherently prone to cause breakage (as we later show in Section 5.4.4) because it blocks access to content or cookies that might be essential for website functionality.

Focus of this work. In conclusion, prior work has only incidentally measured the usage of first-party tracking cookies and existing approaches to detect first-party cookies are lacking. In this chapter, we fill this void by conducting a large-scale study to measure the prevalence of first-party tracking cookies and develop an accurate and robust machine learning approach, called CookieGraph, that is purpose-built to detect first-party

cookies.

5.3 Measurements

In this section, we present a measurement study to understand the usage of first-party cookies by advertising and tracking services (ATS) when third-party cookies are blocked. To this end, we conduct two web crawls (with and without third-party cookies) and analyze the differences in the tracking activity (i.e., sharing of identifiers to known adverting and tracking services) observed across these two crawls to understand the effectiveness of third-party cookie blocking and whether first-party cookies are used in lieu of third-party cookies.

5.3.1 Data Collection and Methodology

Data collection. We use OpenWPM [117] to crawl sites from Alexa's top-100K list. To ensure that our crawls contain representative sites of different popularity, we crawl the top 1K sites, and randomly sample another 9K sites from the long tail of sites ranked 1K-100K. To ensure intra-page diversity (landing and internal pages [233]) we perform an interactive crawl. Specifically, for each site, we crawl its landing page, and then sample 5-10 anchor tags in this landing page uniformly at random, and crawl them to get a sample of internal pages. We conduct two crawls: one with third-party cookies enabled (`3P-Allowed`), and one with third-party cookies blocked (`3P-Blocked`). We conduct these crawls simultaneously to minimize temporal variations in sites across the two crawls[1].

Definition of first- and third-party cookies. Cookies are set in the browser in two ways. They can either be set by the Set-Cookie HTTP response header or by using document.cookie() in JavaScript. Cookies are further classified as first- or third-party. Cookies set via response header from the same (or different) domain as the first-party are first-party (or third-party) cookies. Classification of cookies set by a script depends on whether the script is embedded in a first- or third-party execution context. The cookies set by third-party scripts running in a first-party context are first-party cookies. The cookies set by third-party scripts running in a third-party context (e.g., third-party iframes) are third-party cookies.

Labeling tracking activity. We use EasyList [131] and EasyPrivacy [132] to label requests as tracking (ATS) or not tracking (Non-ATS).[2] Since the basic premise of tracking

[1]The success rate of our crawl is 83.98%. Form the 10K sites visited, 8,398 were successfully crawled.

[2]We label a request as tracking (ATS) if its URL matches the rules in either one of the lists. Otherwise, we label it as not tracking (Non-ATS).

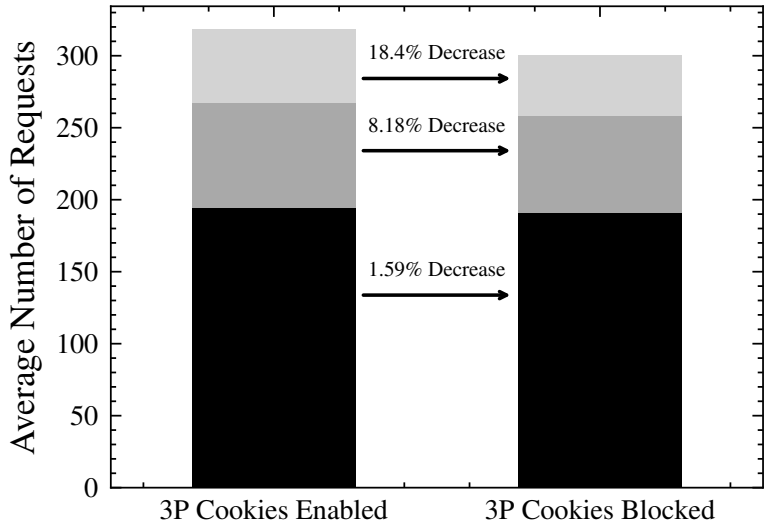


Figure 5.1: Average number of requests per site in `3P-Allowed` and `3P-Blocked` configurations:
(▬) Non-ATS requests
(▬) ATS requests without identifiers
( ) ATS requests with identifiers

is to identify users, we are particularly interested in sharing of identifiers in these tracking requests. To this end, in line with prior work [234, 235], we define identifiers as a string that is longer than 8 characters and matches the regex $[a-zA-Z0-9\_=-]$. Using this definition, we look for identifiers in URL query parameters [236] and cookie values [195, 237, 199, 208].

5.3.2 Tracking after Blocking Third-Party Cookies

We first study whether blocking third-party cookies effectively eliminates ATS requests. To this end, we compare the number of requests with and without third-party cookies.

Figure 5.1 plots the number of requests with and without third-party cookies. It can be seen from the Figure 5.1 that when third-party cookies are blocked, there is only a modest reduction in the overall number of ATS requests, with just an 18.4% reduction in the number of ATS requests containing identifiers. This is surprising because cookie syncing, which is widely used for cross-site tracking [148, 146], entails sharing third-party identifier cookies in query parameters [237, 199, 208]. With third-party cookies blocked, cookie syncing between third-parties cannot occur and we would expect to see a larger drop in identifiers shared in ATS requests. We address this surprising observation in Section 5.3.3.

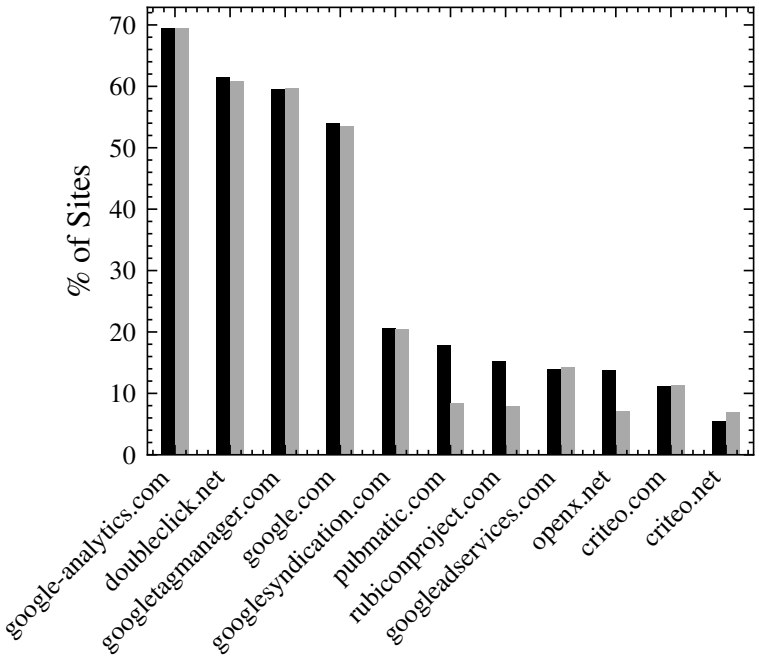


Figure 5.2: Presence of top-10 tracking domains. The plot shows percentage of sites where at least one request is sent to a tracking domain. We include `criteo.net` due to its peculiar increased presence after blocking third-party cookies.
(▬) third-party cookies allowed
(▬) third-party cookies blocked

Next, we analyze whether third-party cookie blocking disparately impacts different ATS domains (eTLD+1). Figure 5.2 plots the percentage of sites with at least one ATS request with identifiers for the top-10 most prevalent ATS domains across both crawls. We note that six of the top-10 ATS domains, all owned by Google, show only a negligible reduction in the number ATS requests with identifiers when third-party cookies are blocked. In contrast, three other ATS domains, owned by Pubmatic, Rubicon, and OpenX, show an almost 50% reduction. Criteo exhibits interesting behavior, where the requests sent to Criteo are divided between two domains: criteo.com and criteo.net. While criteo.com shows negligible change across two crawls, criteo.net increases by about one-third. We attempt to better understand the reason behind this disparate impact across different ATS domains.

5.3.3 Tracking through First-Party Cookies

Figure 5.1 showed that 82.6% of ATS requests contain identifiers even after third-party cookies are blocked. It is clear that the identifiers in these ATS requests are then likely originating from some storage mechanism other than third-party cookies. Since recent

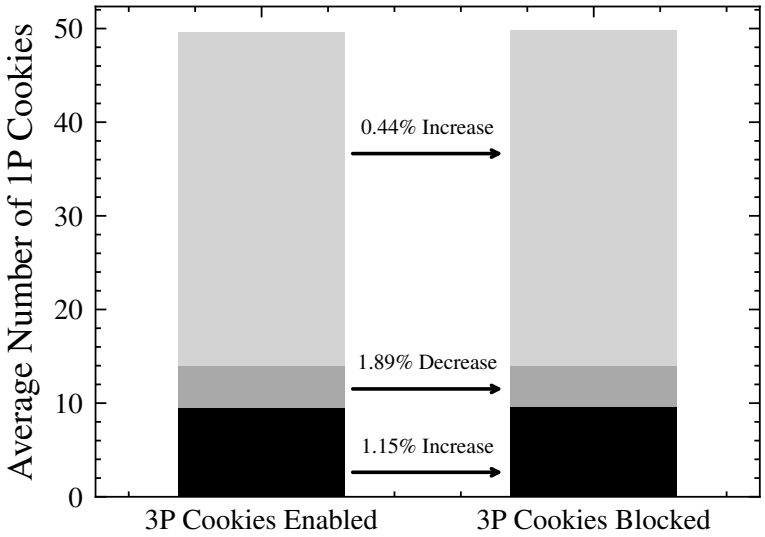


Figure 5.3: Breakdown of average number of first-party cookies per site set before and after blocking third-party cookies.
(▬) Cookies set by non-tracking sources
(▬) Non-identifier cookies set by tracking sources
( ) Identifier cookies set by tracking sources.

prior work has shown that ATSes are increasingly using first-party cookies [195, 199], we next investigate whether first-party cookies are being used in lieu of third-party cookies.

We first compare the average number of first-party cookies in `3P-Allowed` and `3P-Blocked` crawls in Figure 5.3. We observe only a minor difference in the average number of first-party cookies set by ATS scripts (or Non-ATS for that matter).[3] However, it is noteworthy that 81% of the first-party cookies are set by ATS scripts and further 82% of them are identifier cookies. This demonstrates that an overwhelming number of first-party cookies are in fact set by ATSes.

Next, we compare the setting of first- and third-party identifier cookies by ATS domains (eTLD+1 of the setting script URL) to understand if first-party cookie usage is equally prevalent across different ATSes. Figure 5.4 plots the percentage of sites where at least one first-party and/or third-party identifier cookie is set by top-10 ATS domains (with Criteo divided into criteo.com and criteo.net). First, we observe that for the six Google-owned ATS domains, which showed negligible difference in requests containing identifiers after blocking third-party cookies, there is also little to no change in use of first-party identifier cookies across both crawls. These domains do not set a large number of third-party

[3]We label scripts as ATS or Non-ATS based on their `src` URL as in Section 5.3.1.

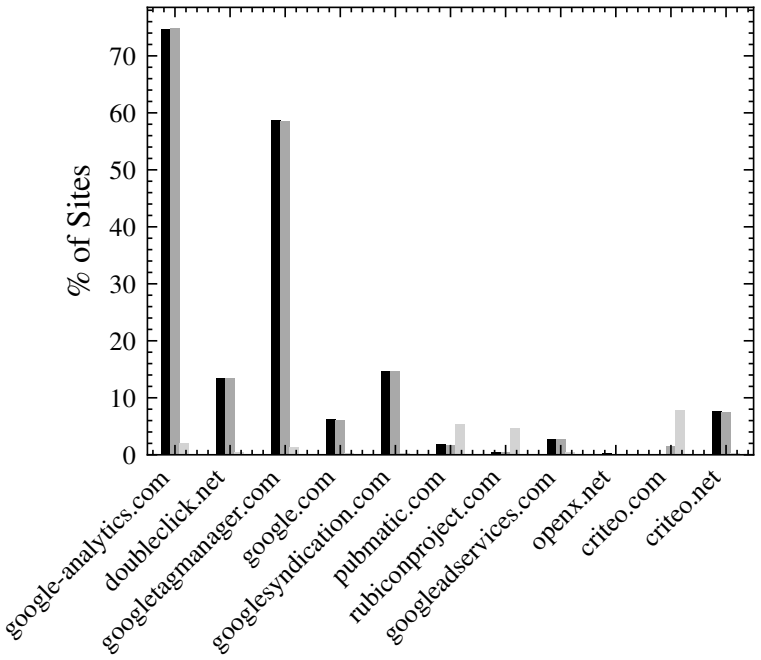


Figure 5.4: Comparison of percentage of sites on which first party and third-party identifier cookies are set by ATS domains.
(▬) first-party cookies set when third-party cookies are allowed
(▬) first-party cookies set when third-party cookies are blocked
(▬) third-party cookies set when third-party cookies are allowed

identifier cookies, which likely explains why they were not impacted by third-party cookie blocking. Second, the other set of ATS domains (i.e., Pubmatic, Rubicon, and OpenX) disproportionately use more third-party identifier cookies than first-party identifier cookies. This observation explains the drastic drop in number of requests containing identifiers to these other ATS domains after blocking third-party cookies in Figure 5.2.

Finally, we further investigate Criteo which showed a peculiar behavior in Figure 5.2. Recall that Criteo uses criteo.com and criteo.net to set cookies: the first-party identifier cookies set by the former showed an increase after blocking third-party cookies while the latter does not change. In addition to this, criteo.com is also used to set third-party identifier cookies, while criteo.net sets only first-party identifier cookies. Both of these domains set the same `cto_bundle` cookie. To compare `cto_bundle` with identifier cookies set by other ATSes, we plot the percentage of sites where a cookie with the same name appears. Figure 5.5 plots the prevalence of first-party cookies for top-20 cookies. We note that while other cookies witness a slight drop in their prevalence after blocking third-party cookies, it does not hold true for `cto_bundle`. In fact, `cto_bundle`'s prevalence increased after blocking third-party cookies, in accordance with the unexpected increase in total

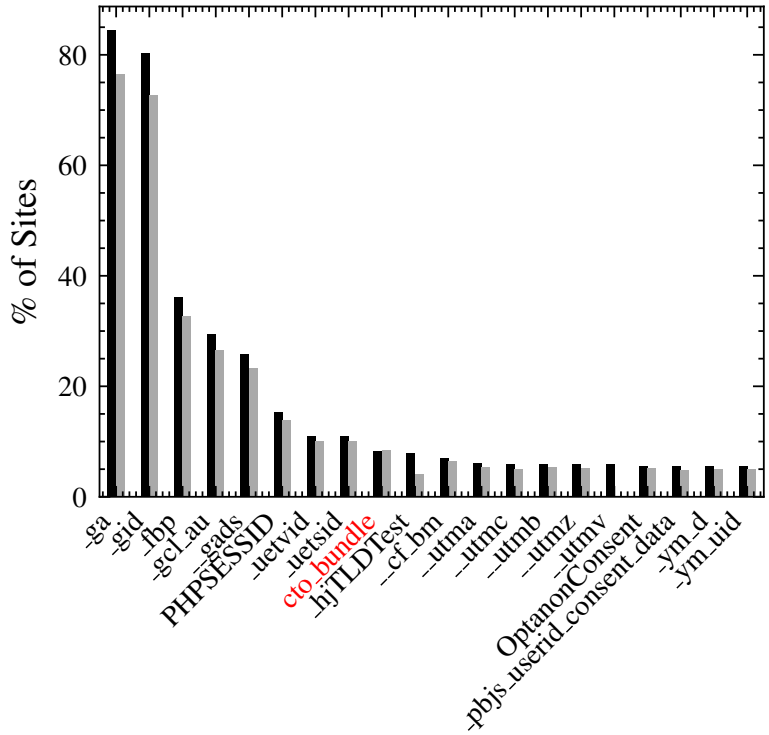


Figure 5.5: Percentage of sites first-party cookies show up on before and after blocking first-party cookies.
(▬) third-party cookies are allowed
(▬) third-party cookies are blocked.

number of first-party identifier cookies set by scripts belonging to criteo.com.

The aforementioned increased use of first-party cookies represents an interesting scenario where trackers are reactively shifting to first-party cookies if third-party cookies are blocked. We can quantify this shift as a ratio between the number sites where first-party cookie is present in `3P-Allowed` crawl and number of new sites where first-party cookie is present in `3P-Blocked` crawl. Table 5.1 shows top-10 identifier cookies based on this ratio. We note that the list is dominated by three well-known ad-tech organizations: Criteo (`cto_bundle`), ID5 (`id5id`, `pbjs-unifiedid`, `pbjs-id5id`), and Lotame (`panoramaID`, `_cc_id`). We further investigate the behavior of these cookies using their publicly available documentation [238, 239, 240, 241, 242] in Appendix D.1.

5.3.4 Cross-site Tracking via First-party Cookies

Our analysis of Criteo, Lotame, and ID5 in Appendix D.1 reveals a common approach to using first-party cookies for cross-site tracking. They build an "identity graph" to

Table 5.1: Cookies showing the highest ratio of new appearances after blocking 3P cookies. The ratio is calculated by dividing the new appearances of a cookie after blocking third-party cookies by total appearances of that cookie before blocking third-party cookies.

Cookie Name	Script Domain	New Appearances	Total Appearances	R
cto_bundle	criteo.com	132	632	0.21
id5id	pubmatic.com	25	139	0.18
pbjs-unifiedid	pubmatic.com	17	103	0.16
pbjs-id5id	pubmatic.com	24	164	0.15
s_sq	adobedtm.com	17	122	0.14
__stripe_mid	stripe.com	13	95	0.14
__stripe_sid	stripe.com	13	95	0.14
panoramaId	crwdcntrl.net	24	184	0.13
_cc_id	crwdcntrl.net	24	196	0.12

link different identifiers to a particular user using first-party information collected from different sites. A node can represent a user (or device such as web browser) based on

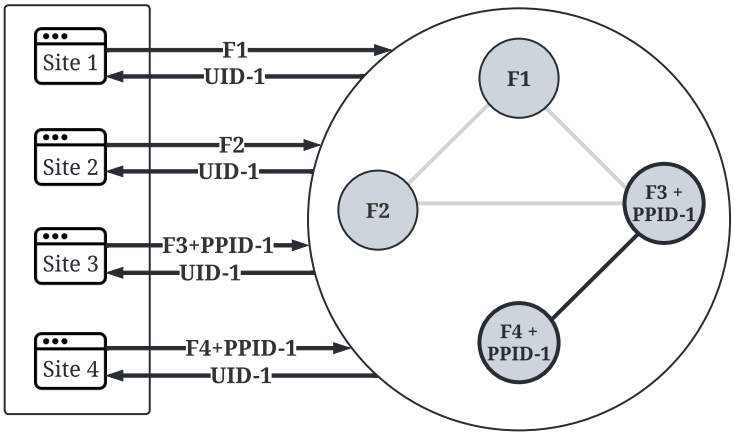


Figure 5.6: The figure shows the flow of information and identifiers through an identity Graph for cross-site attribution. Initially, the user visits sites 1, 2, and 3. Trackers on sites 1, 2, and 3 collect and send fingerprints $F1$, $F2$, and $F3$ to their identity graph. The identity graph returns a $UID-1$ for all the site visits, using a probabilistic matching of fingerprints $F1$, $F2$, and $F3$ sent on each respective website. A publisher provided ID, $PPID-1$, is also sent alongside $F3$ when visiting site 3. When the user visits site 4, it sends fingerprint $F4$. Because the fingerprint $F4$ is different from $F1$, $F2$, and $F3$, the identity graph cannot create a probabilistic match with the other sites. On site 4, the website obtains and sends a publisher provided ID which matches $PPID-1$ provided on site 3. As a result, the identity graph matches and returns the existing user's $UID-1$ for site 4 using deterministic matching. All of these IDs are stored in first-party cookies on the user's device.

different attributes and edges between the nodes are formed based on "deterministic" or "probabilistic" matching between attributes of a pair of nodes. For cross-site tracking, they need to establish edges between different nodes (that actually represent the same user/device) of the identity graph. Note that trackers cannot simply use third-party identifier cookies if they are blocked.

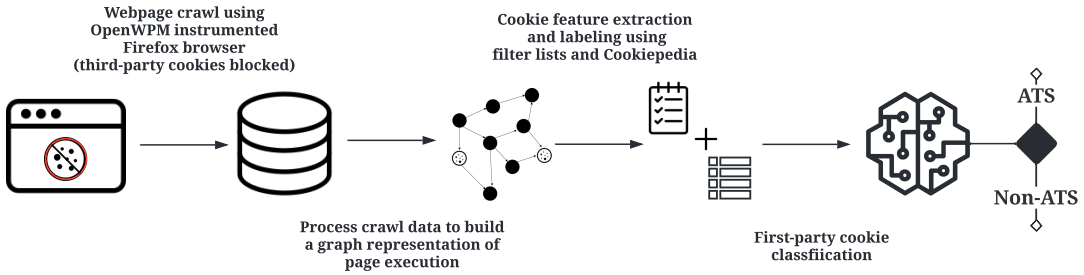


Figure 5.7: Overview of CookieGraph pipeline: (1) Webpage crawl using an instrumented browser; (2) Construction of a graph representation to represent the instrumented webpage execution information; (3) Feature extraction for graph nodes that represent first-party cookies; and (4) Classifier training to detect first-party ATS cookies.

Trackers typically use two types of information to build their identity graph. They gather information provided by publishers including both deterministic attributes (e.g., email, phone, username, or any other publisher-provided ID [PPID] that can be directly used for identification) and probabilistic attributes (e.g., zip code, city, age, etc. that can be used together for non-deterministic identification). They themselves typically also gather probabilistic information such as IP address and fingerprinting attributes such as browser and operating system information (e.g., name and specific version), device properties (e.g., display resolution, screen orientation), etc.

To link different nodes in their identity graph (e.g., to link the same user across different sites or to link different devices of the same user, an example showing how different user devices are linked is shown in Appendix D.2), they use probabilistic or deterministic matching as shown in more detail in Figure 5.6. In probabilistic matching, they measure the similarity between probabilistic attributes and determine a match if the similarity is reasonably high (represented as gray edges in Figure 5.6). In deterministic matching, they can exactly match deterministic attributes (represented as black edges in Figure 5.6). Once these links are established, trackers store an identifier in a first-party cookie which uniquely represents that user across different sites (or devices).

5.3.5 Takeaway

Our differential measurement study reveals that blocking third-party cookies is insufficient in preventing tracking; as there is a minimal decrease in the number of ATS requests sharing identifiers when third-party cookies are blocked. However, the impact of third-party cookie blocking is not uniform across different ATSes– some ATS domains such as google-analytics.com and doubleclick.net show no change in their tracking requests, while others such as pubmatic.com and rubiconproject.com show a decrease, and yet others such as criteo.net show an increase. We find that first-party cookies are predominantly used by ATSes in lieu of third-party cookies to perform tracking. Some ATS domains, such as those owned by Google, only use first-party cookies and are hence not impacted by third-party cookie blocking. Some other ATS domains that do use third-party cookies reactively shift to using first-party cookies when third-party cookies are blocked. We find that these ATSes rely on a combination of deterministic and probabilistic attributes to build an identity graph. Then, they use first-party cookies to store these identifiers that are used for cross-site tracking.

Next, we present our approach to accurately and robustly detect these first-party ATS cookies.

5.4 CookieGraph: Detecting First-Party Tracking Cookies

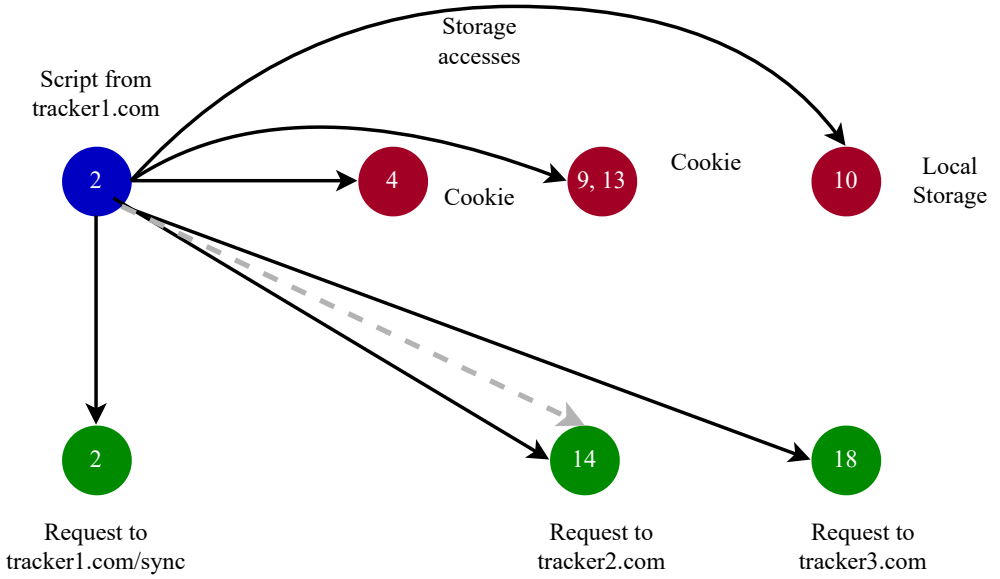
In this section, we describe CookieGraph, a graph-based machine learning approach to detect first-party ATS cookies. CookieGraph creates a graph representation of a webpage's execution based on HTML, network, JavaScript, and storage information collected by an instrumented browser, in which first-party cookies are represented as storage nodes. CookieGraph extracts distinguishing features of these cookies and uses a random forest classifier to detect first-party ATS cookies. Figure 5.7 provides an overview of CookieGraph's pipeline.

5.4.1 Design and Implementation

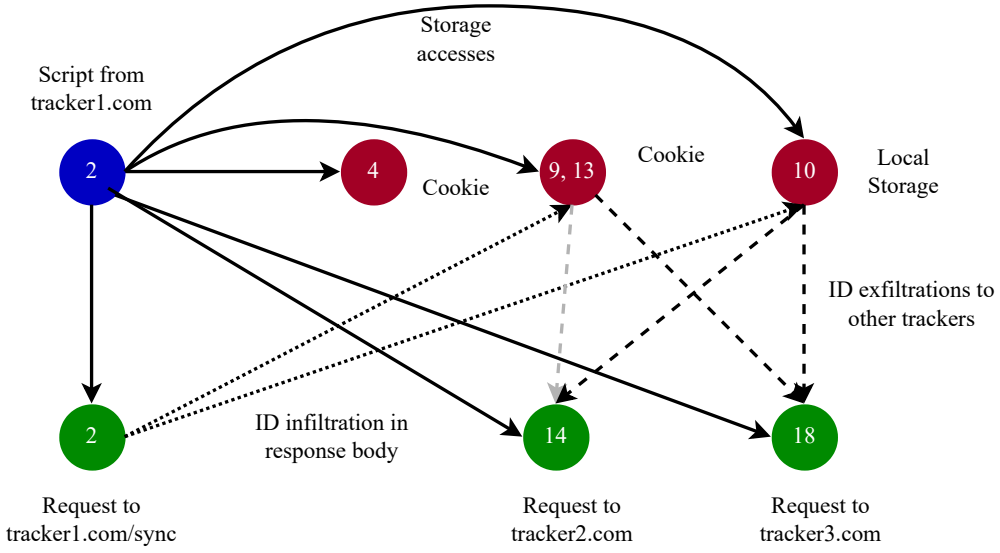
Browser instrumentation. CookieGraph relies on our extended version of OpenWPM [117] to capture webpage execution information across HTML, network, JavaScript, and the storage[4] layers of the web stack. Specifically, CookieGraph captures HTML elements created by scripts, network requests sent by HTML elements (as they are parsed) and

[4]Our measurements in Section 5.3 found a significant use of localStorage in addition to cookies. Thus, we use the term "storage" to refer to both cookies and localStorage. In most cases, the description for cookies is also applicable to localStorage and vice versa.

scripts, responses received by the browser, exfiltration/infiltration of identifiers in network requests/responses, and read/write operations on browser's storage mechanisms.



(a) Graph representation of Code 5.1 in WebGraph



(b) Graph representation of Code 5.1 in CookieGraph

Figure 5.8: Graph representation of Code 5.1 in WebGraph and CookieGraph. ● represents network nodes, ● represents script nodes, and ● represents storage nodes. Node numbers correspond to the lines in Code 5.1. In Figure 5.8a, dashed (- - -) and dotted (. . .) lines represent the additional edges that are captured by CookieGraph and missed by WebGraph. The grey dashed line shows different representations of the same event by both systems.

```
<html>
      <script src='tracker1.com/track.js'>
       ...
       infoCookie = document.cookie;
       var idReq = new XMLHTTPRequest();
       idReq.open("POST", "tracker1.com/sync", true)
       idReq.send(infoCookie)
       var response = newReq.response
       document.cookie = "IDStore=" + response;
       localStorage.setItem(IDStore, response);
       ...
       var exfilReq1 = new XMLHTTPRequest();
       idCookie = document.cookie
       exfilReq1.open("GET", "tracker2.com?user_id=" + idCookie);
       ...
       var exfilReq2 = new XMLHTTPRequest();
       exfilReq2.setRequestHeader("ID-header", idCookie);
       exfilReq2.open("GET", "tracker3.com");
       ...
      </script>
      ...
</html>
```

Code 5.1: Script from third party tracker1.com executing in a first party context. The script obtains a UID from a sync point, stores it, and exfiltrates it to tracker2.com and tracker3.com.

```
-----------------------------------------------------
Request 1
URL: tracker1.com/sync
POST data: publisherID=704; signature=xyz
Response 1
Status: 200
Content: UID=abcd
-----------------------------------------------------
Request 2
URL: tracker2.com?user_id=abcd
Response 2
Status: 200
-----------------------------------------------------
Request 3
Header: ID-header = abcd
URL: tracker3.com
Response 3
Status: 200
```

Listing 5.1: HTTP requests and responses initiated from Code 5.1.

Graph construction. The nodes in CookieGraph's graph represent HTML elements, network requests, scripts, and storage elements. When localStorage and first-party cookie nodes share the exact same name, CookieGraph considers them as one storage node. The edges represent a wide range of interactions among different types of nodes *e.g.*, scripts sending HTTP requests, scripts setting cookies *etc.* In addition to interactions considered by prior work [200], CookieGraph incorporates edges that capture the tracking behavior of first-party cookies. Informed by our findings in Section 5.3: cookies are typically set with the values *infiltrated* with HTTP responses and are *exfiltrated* via URL parameters and request headers or bodies; CookieGraph captures infiltrations and exfiltrations by linking the script-read/write cookies in the first-party execution context to the requests of reader/writer script that contains those cookie values. In addition to plain text cookie values, CookieGraph also monitors Base64-, MD5-, SHA-1-, and SHA-256- encoded cookie values in URLs, headers, request and response bodies. As in our measurement study, because of the focus on identifiers, CookieGraph only captures cookie values that are at least 8 characters long.

We illustrate the difference between CookieGraph's graph representation and prior work, i.e., WebGraph [200] using an example script that involves first-party ATS cookies. Code 4.1 shows a third-party script from tracker1.com executing in a first party context on a webpage. The script first reads `infoCookie`, which stores tracking information such as the publisher ID and a user signature. Then, it sends the content of the cookie to an endpoint via an HTTP POST request. The endpoint returns a user ID (UID) in the response body, which is stored in both a first-party cookie and localStorage named `IDStore`. At a later point, the script exfiltrates UID to two other tracking endpoints: to tracker2.com via a URL parameter and to tracker3.com via an HTTP header. The HTTP requests and responses that result from Code 5.1 are listed in Listing 5.1.

Figure 5.8 shows the differences between the graph representations of this script created by prior work, WebGraph (left), and CookieGraph (right). WebGraph does not capture the infiltration of the UID to the cookie from the response body and also does not consider infiltration and exfiltration via localStorage. In contrast, the dotted and dashed lines in Figure 5.8b show that CookieGraph captures both the infiltration and the exfiltration in subsequent network requests. Moreover, while WebGraph captures exfiltrations via URL parameters (shown by grey dashed lines) via edges from the setting script to the endpoint, CookieGraph is able to precisely *link this exfiltration to the first-party cookie* via an edge from the cookie node to the endpoint.

Feature extraction. We use CookieGraph's representation to extract structural and information flow features.

Table 5.2: CookieGraph features comparison with WebGraph. ● indicates that a feature is present. ◖ indicates that feature was extended in CookieGraph. CookieGraph calculates Graph size, Degree and Centrality features using both normal and shared information edges. The former comes under structural features while the latter comes under flow features.

Feature	Type	WebGraph	WebGraph
Graph size (# of nodes, # of edges, and nodes/edge ratio)	Structure	●	●
Degree (in, out, in+out, and average degree connectivity)	Structure	●	●
Centrality (closeness centrality, eccentricity)	Structure	●	●
Ascendant's attributes	Structure	●	●
Descendant of a script	Structure	●	●
Ascendant's script properties	Structure	●	●
Parent is an eval script	Structure	●	●
Local storage access (# of sets, # of gets)	Flow	●	●
Cookie access (# of sets, # of gets)	Flow	●	●
Storage access on local storage with same name (# of sets, # of gets)	Flow	●	
Requests (sent, received)	Flow		●
Redirects (sent, received, depth in chain)	Flow		●
Common access to the same storage node	Flow		●
Cookie exfiltration	Flow	●	◖
Cookie infiltration	Flow	●	
Cookie Setter (# of exfiltration, # redirects)	Flow	●	
Graph size (# of nodes, # of edges, and nodes/edge ratio)	Flow	●	●
Degree (in, out, in+out, and average degree connectivity)	Flow	●	●
Centrality (closeness centrality, eccentricity)	Flow	●	●

Structural features represent relationships between nodes in the graph, such as ancestry information and connectivity. These features capture the relationships between the first-party cookie nodes and scripts on the page. For example, how many scripts interacted with a cookie or whether a script that interacted with a cookie also interacted with other cookies.

Flow features represent first-party ATS cookie behavior. We extract three types of flow features. First, we count the number of times a cookie was read or written. Second, we count the number of times a cookie was infiltrated or exfiltrated via the methods explained in the previous section. Third, we calculate some features with respect to the setter of the cookie. Concretely, whether the setter's domain also acted as an end-point for other cookie exfiltrations, and whether the setter's domain was involved in redirect chains (since redirects are commonly used in tracking). The intuition behind the third category of features is that domains involved in setting first-party ATS cookies are also involved in sharing information with other ATSes.

Table 5.2 shows the differences in features between CookieGraph and WebGraph. CookieGraph adds improved cookie exfiltration features and also introduces a new complete new set of infiltration and setter features. Unlike WebGraph, CookieGraph also considers cases where a localStorage shares the same name as a cookie (a behavior

observed in first-party ATS cookies). CookieGraph does *not* use content features, e.g., based on cookie name, as they can be trivially used in detection evasion tactics [200, 234]. CookieGraph also removes three features because they are related to classification of request nodes in WebGraph whereas CookieGraph classifies storage nodes.

5.4.2 Evaluation

Similar to previous work [142, 200], we use a random forest classifier to distinguish between ATS and Non-ATS cookies. We first train and test the accuracy of this classifier on a carefully labeled dataset. Then, we deploy it on our 10K website dataset.

Ground truth labeling

We use two complementary approaches to construct our ground truth for first-party ATS cookies. We represent each first-party cookie as a cookie-domain pair, since the same cookie name can occur on multiple sites.

Filter lists. We rely on filter lists [131, 132] as previous work has found them to be reasonably reliable in detecting ATS endpoints [142, 200]. However, filter lists are designed to label resource URLs, rather than cookies. We adapt filter lists to label cookies by assigning the label of a particular resource to all the cookies set by that resource. Since both ATS and Non-ATS cookies can be set by the same resource, this labeling procedure could result in a non-trivial number of false positives. To limit the number of false positives in our ground truth, we only label Non-ATS cookies based on filter lists: i.e., if a script that sets a cookie is not marked by *any* of the filter lists, we label these cookies as Non-ATS. Conservatively, if any one of the filter lists mark the cookie's setter as ATS, we label the cookie as Unknown.

Cookiepedia. Inspired by prior work [202], we use Cookiepedia [243] as an additional source of cookie labels. Cookiepedia is a database of cookies maintained by a well-known Consent Management Platform (CMP) called OneTrust [244, 202]. For each cookie and domain pair, Cookiepedia provides its purpose, defined primarily through the cookie integration with OneTrust. Each cookie is assigned one of four labels: strictly necessary, functional, analytics, and advertising/tracking. As Cookiepedia-reported purposes are self-declared, we adopt a conservative approach: we only label a cookie-domain pair as ATS if a cookie's purpose is declared as advertising/tracking or analytics in a particular domain. If the cookie's declared purpose is strictly necessary or functional, we label the cookie as Unknown, as the cookie might have been, mistakenly or intentionally, mislabeled.

We combine the results of the labeling approaches to obtain a final label for first-party cookies. If both approaches label a cookie as Unknown, its final label is Unknown. If only one of the approaches has a known label, this is the final label. When Cookiepedia marks a cookie as ATS and filter lists mark it as Non-ATS, we give precedence to the Cookiepedia label and assign the final label as ATS because websites are unlikely to self-declare their Non-ATS cookies as ATS.

Using this labeling process, 20,927 out of 78,560 first-party cookies (26.64%) have a known (ATS or Non-ATS) label and the rest are labeled as Unknown. We then observe that cookies set by the same script across two different sites are often labeled ATS in one instance and Unknown in other instance because Cookiepedia does not have data for the latter. As it is unlikely that an ATS script changes purpose across sites, we propagate the ATS label to all instances set by the same script. After this label propagation, 51.76% of the data is now labeled, with 21,875 (53.79%) ATS and 18,786 (46.20%) Non-ATS labels.

Classification

We train and test the classifier on the labeled dataset using standard 10-fold cross validation. We ensure that there is no overlap in the websites used for training and test in each fold. Similar to Section 5.4.1, we limit the classifier to cookies whose value is at least 8 characters long. The classifier has 91.87% precision and 90.59% recall, with an overall accuracy of 91.06%, indicating that the classifier is successful in detecting ATS cookies.

Feature analysis. We conduct feature analysis to understand the most influential features for the classifier. We find that the most influential features are the flow features, which capture cookie exfiltrations, set operations, and redirections by cookie setters. Figure 5.9 shows the distributions for the number of cookie exfiltrations (top) and the number of times a cookie is set (bottom), for ATS and Non-ATS cookies. ATS cookies are much more likely to be exfiltrated than Non-ATS cookies: ATS have a median number of 6 exfiltrations (mean/std is 11.11/15.95) as compared to a median of 0 for Non-ATS (mean/std is 0.62/5.29). Also, ATS cookies tend to be set much more frequently by scripts, with a median of 3 set operations (mean and standard deviation is 4.86±6.99) as compared to 1 for Non-ATS cookies (mean and standard deviation is 2.17±6.08). These findings confirm our conclusions in Section 5.3: first-party ATS cookies are used to store identifiers which are then exfiltrated to multiple endpoints.

Error analysis. We conduct manual analysis of CookieGraph's false positives and false negatives to understand why the approach fails.

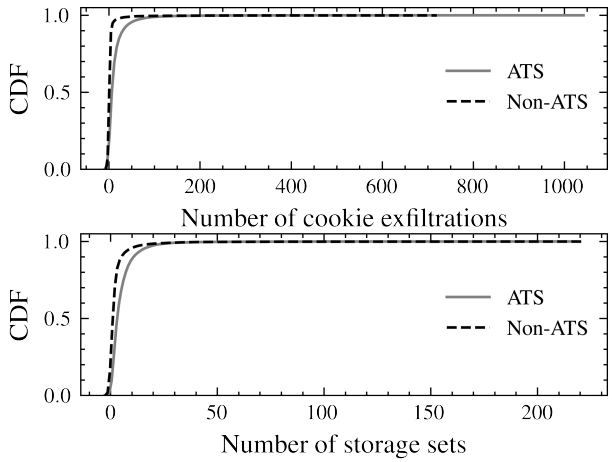


Figure 5.9: Feature distribution of cookie exfiltrations (top) and storage sets (bottom) for ATS and Non-ATS cookies. ATS cookies are exfiltrated and set more than Non-ATS cookies, resulting in flow features based on exfiltrations and sets being helpful for the classifier.

We find that the cookies that were most misclassified as ATS are those whose publicly available descriptions indicate they are used to track visitors on a page (e.g., `__attentive_id`, `messagesUtk`, `omnisendAnonymousID`) [245, 246, 247]. We also find a few instances of well-known Google Analytics cookies `_ga` and `_gid` that are labeled in ground truth as Non-ATS, but are classified by CookieGraph as ATS. Overall, we find that the false positives are typically not caused by CookieGraph misclassifying non-tracking cookies, but mostly that the tracking cookies flagged by CookieGraph were mislabeled as Non-ATS in the ground truth. In other words, CookieGraph has likely correctly classified these tracking cookies. We note that even after our procedures to improve ground truth labels, there may be cookies that did not have self-disclosed labels or were served from slightly different scripts (thereby missing our hash-based script matching) leading to some mistakes in the ground truth. We leave investigation of further methods of improving the ground truth labeling to future work.

For false negatives, a representative case is the `_pin_unauth` cookie. Its value is double-base64-encoded, that is not included in the list of potential encoding schemes used by CookieGraph to detect exfiltration. These false negatives can be averted by using a more comprehensive list of encoding schemes or by performing full-blown information flow tracking instead of approximating exfiltration flows; however, the latter would come at a performance cost as we discuss further in Section 5.4.4. Other false negatives are because CookieGraph does not capture sufficient activity during webpage execution. We further discuss these cases of false negatives in Section 5.5.1

5.4.3 Deployment

We deploy CookieGraph to classify all cookies, including Unknown cookies, in our crawl of 10K sites.

Prevalence of first-party ATS cookies. Overall, CookieGraph classifies 62.48% of the 74,003 first-party cookies in our dataset as ATS. We find that 93.43% of sites deploy at least one first-party ATS cookie. Of these sites, the average number of first-party ATS cookies on a site is 6.29.

Who sets first-party ATS cookies? The vast majority (98.39%) of the first-party ATS cookies are in fact set by third-party embedded scripts served from a total of 1,588 unique domains. This demonstrates that first-party ATS cookies are actually set and used by third-party trackers. Because this is only possible if the first-party allows the third-party trackers to embed a script in first-party context, this suggests that there is intentional or unintentional collusion between the first-party and third-party tracker. These third-party-set first-party cookies enable third-parties to circumvent blocking-based countermeasures implemented by browsers.

Next, we analyze the most prevalent first-party cookies and the third-party entities that actually set them. Table 5.3 lists top-25 out of 5,019 first-party ATS cookies[5] based on their prevalence. Two major advertising entities (Google and Facebook) set first-party ATS cookies on approximately a third of all sites in our dataset. CookieGraph detects `_gid` and `_ga` cookies by Google Analytics as ATS on 77.11% and 68.88% of the sites. The public documentation acknowledges using these two first-party cookies to store user identifiers for tracking [248]. CookieGraph detects `_fbp` cookie by Facebook as ATS on 33.22% of the sites. Their public documentation acknowledges that Facebook tracking pixel stores unique identifier in the first-party `_fbp` cookie [249]. In fact, Facebook made a recent change to include first-party cookie support in its tracking pixel to avoid third-party cookie countermeasures [250].

TikTok, an emerging social media app that is known to aggressively harvest sensitive user information [251], also recently added support for setting first-party tracking cookies using TikTok Pixel [252, 253]. TikTok's first-party `_ttp` tracking cookie is present on 3.75% percent of sites, which is considerably lower than Facebook and Google but comparable to more specialized entities such as Criteo.

Criteo's `cto_bundle` cookie is amongst the most prevalent first-party ATS cookies. Recall from Section 5.3.3 that `cto_bundle` is sometimes purposefully set when third-party cookies

[5]We report distinct tuples of cookie name and the setter script's URL.

are blocked. Our deployment of CookieGraph shows that Criteo sets this first-party ATS cookie on 5.98% of sites in our dataset. Note that first-party ATS cookies from Lotame, ID5, and Adobe listed in Table 5.1 are also detected by CookieGraph but they do not make the top-25 list. Despite not being as prevalent as the other first-party ATS cookies, their behavior analysis in Section 5.3.3 was crucial in discovering prevalent examples discussed in this section.

Table 5.3: List of top-25 ATS cookies detected by WebGraph

Cookie Name	Script Domain	Org.	Percentage of Sites
_gid	google-analytics.com	Google	77.11%
_ga	google-analytics.com	Google	68.88%
_fbp	facebook.net	Facebook	33.22%
_gcl_au	googletagmanager.com	Google	14.22%
__gpi	googlesyndication.com	Google	14.02%
_ga	googletagmanager.com	Google	12.79%
__gads	googlesyndication.com	Google	12.35%
__gads	doubleclick.net	Google	11.68%
_uetsid	bing.com	Microsoft	10.22%
_uetvid	bing.com	Microsoft	10.22%
__gpi	doubleclick.net	Google	10.11%
_clck	clarity.ms	Microsoft	8.81%
_hjTLDTest	hotjar.com	Hotjar	8.05%
_clsk	clarity.ms	Microsoft	7.88%
cto_bundle	criteo.net	Criteo	5.98%
_ym_d	yandex.ru	Yandex	4.85%
_ym_uid	yandex.ru	Yandex	4.85%
_pin_unauth	pinimg.com	Pinterest	4.57%
__utma	google-analytics.com	Google	4.32%
__utmb	google-analytics.com	Google	4.32%
__utmz	google-analytics.com	Google	4.32%
__qca	quantserve.com	Quantcast	4.19%
__utmc	google-analytics.com	Google	4.17%
_ttp	tiktok.com	TikTok	3.75%
hubspotutk	hs-analytics.net	HubSpot	3.29%

Browser fingerprinting. As discussed in Section 5.3.4, trackers that use first-party ATS cookies may employ other invasive tracking techniques such as browser fingerprinting to implement cross-site tracking. We analyze the first-party cookies that are set by the scripts from entities that are known to engage in browser fingerprinting. We use Disconnect's sub-list of fingerprinters [254, 255] from its tracking protection list [220]. We find that

Google's and Facebook's first-party ATS cookies are predominately set by scripts served from domains involved in fingerprinting. Lotame's cookies (`_cc_id`, `_cc_aud`, `_cc_cc`) are also found to be set by such scripts.

Overall, we find that 45 (2.83%) distinct domains that set first-party cookies are also known fingerprinters. However, these handful of domains are responsible for setting 41.45% of all first-party ATS cookies. This disproportionately between domains and number of cookies set is not surprising. Effective cross-site tracking would require a tracker to be present on and collect data from a large number of sites. This presence will allow the tracker to collect extensive deterministic and probabilistic attributes about the user from a varied number of source, enhancing its ability to track users across sites in absence of third-party cookies. Our case studies in Appendix D.1 and our analysis in Section 5.3.4 elaborate on how first-party ATS cookies are combined with fingerprinting for cross-site tracking.

5.4.4 Comparison with Existing Countermeasures

Next, we compare CookieGraph with state-of-the-art countermeasures against ATS, CookieBlock [202] and WebGraph [200], in terms of detection accuracy, website breakage, and robustness.

CookieBlock is a state-of-the-art approach to classify cookies, including advertising/-tracking and analytics. It makes use of both manually curated allow lists and a machine learning classifier, which mainly relies on features based on cookie attributes (cookie names and values).

WebGraph is the state-of-the-art graph-based approach to classify ATS requests. Since WebGraph is not designed to directly classify cookies, we adapt it to this end by identifying ATS resources identified by WebGraph in `3P-Blocked` and generating a block list of cookies for each domain set by those resources. This list is meant to mimic the effect of blocking these resources on first-party ATS cookies.

Detection Accuracy

Table 5.4 compares the detection accuracy of CookieGraph with CookieBlock and WebGraph. CookieGraph outperforms both approaches in all metrics. The superiority in precision indicates that existing countermeasures result on many more false positives than CookieGraph. These additional false positives means that previous approaches would block functional first-party cookies potentially affecting user experience. Next, we investigate the impact of these false positives on website breakage.

Table 5.4: Classification accuracy of CookieGraph, WebGraph, and CookieBlock

Classifier	Accuracy	Precision	Recall
CookieGraph	91.06%	91.87%	90.59%
WebGraph	78.74%	71.59%	85.49%
CookieBlock	80.78%	69.95%	72.45%

Website Breakage

We manually analyze the breakage caused by CookieGraph, CookieBlock and WebGraph's on 50 sites that are sampled from the 10K sites used in Section 5.3 (25 sites chosen randomly from top 100 and other 25 from the rest).

We divide our breakage analysis in four categories of typical website usage: navigation (from one page to another), SSO (initiating and maintaining login state), appearance (visual consistency), and miscellaneous functionality (chats, search, shopping cart, etc.). We label breakage as major or minor for each category: major breakage – when it is not possible to use a functionality on the site included in either of the aforementioned categories, and minor breakage – when it is difficult, but not impossible, for the user to make use of a functionality. To assess website breakage, we compare a vanilla Chrome browser (with no countermeasures against first-party cookies) with browsers enhanced with an extension which blocks all first-party cookies classified as ATS by CookieGraph, enhanced with an extension which blocks all cookies set by resources labeled as ATS by WebGraph, and enhanced with the official CookieBlock extension [256]. We use two reviewers to perform the breakage analysis to mitigate the impact of biases or subjectivity. Any disagreements between the reviewers were resolved after careful discussion.

Out of the 50 sites, CookieGraph only had minor breakage on one site where an offer popup kept reappearing due to deletion of a cookie which stores user preferences. In contrast, both WebGraph and CookieBlock cause major breakage in at least one of the four categories on 10% of the sites. For example, WebGraph causes issues with cart functionality on darsoo.com, complete website breakage on espncricinfo.com, and SSO issues on other sites. Most of the breakage issues of CookieBlock relate to SSO logins and additional login-dependent functionality (e.g., missing profile picture). Our results, that CookieBlock causes breakage on 8% of the sites with SSO logins, are inline with the 7-8% breakage reported by the authors [202].

We also find that WebGraph blocks some additional first-party cookies that are important for server-side functionality, but not directly related to user experience and therefore not immediately perceptible. For example, WebGraph blocks essential cookies such as `Bm_sz`

Table 5.5: Website breakage comparison of all three countermeasures.( ) signifies no breakage, ( ) minor breakage, and ( ) major breakage. Each cell represents the percentage of sites on which breakage was observed.

Classifier	Navigation		SSO		Appearance		Miscellaneous	
	Minor	Major	Minor	Major	Minor	Major	Minor	Major
CookieGraph	2%	0%	0%	0%	0%	0%	0%	0%
WebGraph	4%	2%	0%	6%	0%	2%	4%	10%
CookieBlock	0%	0%	0%	8%	0%	2%	0%	2%

cookie used by Akamai for bot detection, `XSRF-TOKEN` cookie used to prevent CSRF on different sites, and `AWSALB` cookies used by Amazon for load balancing. CookieGraph correctly classified these cookies at Non-ATS, and thus does not prevent these measures from being deployed.

Robustness

We compare the robustness of CookieGraph, CookieBlock, and WebGraph to evasion, i.e., modifications to cause the misclassification of ATS resources as Non-ATS. Since advertisers and trackers are known to engage in the arms race with privacy-enhancing tools [158, 151, 147], it is important to test whether the detection of first-party ATS cookies is brittle in the face of trivial manipulation attempts such as changing cookie names.

We evaluate robustness on a test set of 2,000 sites from our dataset which also have the required CMP needed by CookieBlock for data collection and training. This translates to a total test set of 7,726 first-party cookies. We change the names of the cookies in our test set to randomly generated strings of lengths between 2 and 15 characters. Table 5.6 shows the results. We note that both CookieGraph and WebGraph are fully robust to manipulation of cookies names while CookieBlock's accuracy degrades by more than 15%. CookieGraph and WebGraph are robust because they do not use any content features (features related to the cookie characteristics, such as cookie name or domain) since these can be somewhat easily manipulated by an adversary aiming to evade classification [200]. On the contrary, the most important feature of CookieBlock in fact depends on the cookie name, i.e., whether the name belongs to the top-500 most common cookie names [257]. Thus, CookieBlock can be easily bypassed with trivial cookie name modifications.

CookieGraph's implementation of flow features can be manipulated by an adversary by using a different encoding than it currently considers or by changing the domains of exfiltration endpoints. CookieGraph's robustness to these attacks can be improved

Table 5.6: Robustness (difference in classification accuracy)

Classifier	Δ Accuracy	Δ Precision	Δ Recall
CookieGraph	0.00%	0.00%	0.00%
WebGraph	0.00%	0.00%	0.00%
CookieBlock	-15.68%	-15.08%	-16.54%

by more comprehensive information flow tracking. However, full-blown information flow tracking would incur prohibitively high run-time overheads (up to 100X-1000X [174]) and implementation complexity in the browser [175, 176, 186, 187].

5.5 Limitations

5.5.1 Completeness

CookieGraph relies on a graph representation of interactions between different elements during webpage execution. The completeness of the interactions captured in the graph depends on the intensity and variety of user activity on a webpage (e.g., scrolling activity, number of internal pages clicked). In other words, it is possible that CookieGraph may not detect certain ATS cookies if its graph representation has not captured the interactions between different elements due to insufficient user activity.

To study the impact of user activity on CookieGraph, we recrawl sites performing two to three times more internal page clicks than in the original crawl. We specifically recrawl 238 sites where Criteo's `cto_bundle` cookie was originally classified as Non-ATS by CookieGraph. CookieGraph's deployment on the recrawled sites results in successful detection of Criteo's `cto_bundle` cookie as ATS on 121 of the 238 recrawled sites. We find that the average number of infiltrations (exfiltrations) increase from 1.54 to 2.95 (1.13 to 4.01) across the original and recrawled sites. We surmise that while there are cases where CookieGraph incorrectly classifies ATS as Non-ATS due to incompleteness of the graph representation, its decision reflects the behavior of the cookie at the time of classification. As more interaction is captured in the graph, CookieGraph is able to correctly switch the label to ATS. Moreover, CookieGraph never switch labels from ATS to Non-ATS due to increased interaction. We observed a similar trend for other prevalent first-party ATS cookies in our dataset.

5.5.2 Deployment

CookieGraph's implementation is not suitable for runtime deployment due to the performance overheads associated to the browser instrumentation and machine learning

pipeline. We envision CookieGraph to be used in an offline setting: (1) first-party ATS cookie-domain pairs are detected using machine learning classifier and (2) the detected cookie-domain pairs are added to a cookie filter list such as those already supported in privacy-enhancing browser extensions such as uBlock Origin [258] for run-time blocking. We argue that a reasonably frequent (e.g., once a week) deployment of CookieGraph on a large scale would be sufficient in generating and keeping the filter list up-to-date. While advertisers and trackers can in theory change cookie names at a rate faster than CookieGraph's periodic deployment, updating cookie names frequently is challenging in practice because setting these first-party ATS cookies across many different sites requires tight coordination between different entities. To illustrate the practical issues associated with changing cookie names, consider the legacy `demdex` cookie set by Adobe's embedded script that is then exfiltrated to the demdex.net domain. Adobe's documentation explains that it is difficult to change the legacy name because "... it is entwined deeply with Audience Manager, the Adobe Experience Cloud ID Service, and our installed user base" [259, 260]. If advertisers or trackers are somehow able to overcome these practical challenges and change cookie names at a much faster pace, CookieGraph's online implementation for run-time cookie classification would be necessary. Further research is needed for efficient and effective online implementation of CookieGraph.

5.6 Conclusion

We conducted a large scale differential measurement study to investigate how trackers abuse first-party cookies to circumvent third-party cookie blocking. Our proposed CookieGraph was able to accurately and robustly block first-party tracking cookies, and significantly outperforming the state-of-the-art. Using CookieGraph, we found evidence of widespread abuse of first-party cookies on more than 93% of the tested websites by 1500+ distinct tracking domains, which included major advertising entities such as Google as well as many specialized entities such as Criteo.

6 Conclusion

In this dissertation, we have shown that metadata analysis can be a powerful tool for not only understanding how privacy can be compromised but also for understanding how it can be protected. Through our work on metadata analysis for network privacy, we discovered that network protocols, such as encrypted DNS and QUIC are vulnerable to website fingerprinting. We also found that padding-based countermeasures (as defined in the standards of these protocols) are inadequate against website fingerprinting. Our work on metadata analysis for web privacy has shown that we need to consider ATS-behavior metadata when building robust detection systems. A behavior-based approach can be applied to new tracking techniques that ATS employ to overcome browser countermeasures.

Our research has led to modifications of encrypted DNS systems that are widely deployed by major browser vendors and that service millions of people. We performed responsible disclosure of our findings to Google and Cloudflare, as our findings showed that their DNS resolvers were vulnerable to these attacks. Following our disclosure, Cloudflare activated the padding of the responses from their resolver; this increased their protection against the attacks uncovered in the study and improved privacy for their user base. We presented our work on encrypted DNS and QUIC at the IETF Privacy Enhancements and Assessments Research Group (PEARG) meetings, where the standardization community became interested. Furthermore, considering the interest shown by the ad-blocking community, we presented our work on ATS-blocking at the AdBlocker Developer Summit and at Brave Research. We have also released the code for our published work (Chapter 2 and Chapter 4) [261, 262], and intend to release the code for the other chapters upon publication.

6.1 Future Work

This dissertation opens up potential avenues for future work:

Impact of Application Layer Choices on Website Fingerprintability.

Although our work paints a bleak picture of the current status of countermeasures against website fingerprinting, it also sheds light on what we need to do in order to develop effective defenses. We see that, to truly solve the issue of website fingerprinting, we might need to fundamentally alter how resources are served on the Internet. First, there might be a need for coordination among all the parties that serve the content of a web page. Second, resources might need to be served in a standardized order to minimize differences among pages. Third, these changes should not adversely affect user experience. Building better tools to study the effect of resource packaging and ordering, deciding which entities are best placed to deploy defenses, and understanding whether standardization efforts could assist in countermeasure deployment are promising research directions.

Deployable Systems for ATS Prevention.

Our work shows that it is possible to build ML-based ATS-blocking systems that are also robust against evasion. There are a few challenges that we need to tackle before these systems are practical enough to see deployment in the real world.

First, any ATS-prevention system needs to minimize breakage, i.e., the loss of legitimate website functionality that occurs due to ATS blocking. Identifying and reducing breakage requires significant manual effort from filter-list maintainers at the moment [263, 264]. Although recent work has shown that it is possible to build an automated breakage detection system [265], this is restricted to certain types of breakage and certain types of pages. Studying user-reported examples of breakage to construct a taxonomy, and using this taxonomy to better understand breakage would help us build systems that can automatically detect more forms of breakage. This, in turn, could encourage the deployment of ML-based blocking among developers of ATS-blocking solutions.

Second, ML-based methods to detect ATS and enhance filter lists still rely on filter lists as ground truth for model training, which is turn rely on manual rule creation by maintainers. This leads to a circular dependency between these models and filter lists. While ML approaches might reduce the manual effort, research on alternative approaches to supervised learning, along the lines of unsupervised learning [266] or reinforcement learning [267] could be promising avenues to reduce the dependency on manual rule creation for model training.

Third, we contributed to ATS prevention in the form of filter lists by crawling web pages and analysing ATS a posteriori. ATS prevention, via filter lists, makes it easier for ATS to avoid detection by evolving at a rate faster than filter list updates. To overcome this shortcoming, detection methods need to be able to block ATS on the fly, i.e., *online* (note that the existing solutions that instantly block ATS are not robust). This poses a chicken-or-egg problem where we would like to prevent an ATS event before it occurs, yet we need to observe this event in order to flag it for prevention in the first place. Another issue is that online systems need to capture all information flows without significant overhead. Developing robust online ATS blocking is an area that demands further exploration.

Fourth, the ATS-prevention system needs to be quick to respond to changes in ATS behavior. Our research, thus far, has uncovered that the fundamental behaviors of ATS have proven to be very effective features for use by detection and classification systems. Developing detection techniques that can rapidly adapt to the constantly evolving ATS behavior is a significant problem.

Finally, the insights from our work could potentially be applied to other platforms, such as mobile and IoT, that are also seeing an uptick in ATS [1].

A Appendix for Chapter 2

A.1 Datasets

Table A.1 provides an expanded version of the dataset overview.

Table A.1: Overview of datasets (expanded).

Name	Identifier	Location	Resolver	Client	Platform	# webpages	# samples
Desktop (Location 1)	LOC1	Lausanne	Cloudflare	Cloudflare	Desktop	1,500	200
Desktop (Location 2)	LOC2	Leuven	Cloudflare	Cloudflare	Desktop	1,500	60
Desktop (Location 3)	LOC3	Singapore	Cloudflare	Cloudflare	Desktop (AWS)	1,500	60
Raspberry Pi	RPI	Lausanne	Cloudflare	Cloudflare	Raspberry Pi	700	60
Firefox with Google resolver	GOOGLE	Leuven	Google	Firefox	Desktop	700	60
Firefox with Cloudflare resolver	CLOUD	Leuven	Cloudflare	Firefox	Desktop	700	60
Firefox with Cloudflare client	CL-FF	Leuven	Cloudflare	Cloudflare	Desktop	700	60
Open World	OW	Lausanne	Cloudflare	Cloudflare	Desktop	5,000	3
DoH and web traffic	WEB	Leuven	Cloudflare	Cloudflare	Desktop	700	60
DNS over Tor	TOR	Lausanne	Cloudflare	Cloudflare	Desktop	700	60
Cloudflare's EDNS0 padding implementation	EDNS0-128	Lausanne	Cloudflare	Cloudflare	Desktop	700	60
Recommended EDNS0 padding	EDNS0-468	Lausanne	Cloudflare	Cloudflare	Desktop	700	60
EDNS0 padding with ad-blocker	EDNS0-128-adblock	Lausanne	Cloudflare	Cloudflare	Desktop	700	60
DoT with Stubby client	DOT	Lausanne	Cloudflare	Stubby	Desktop	700	60

A.2 Performance metrics.

In this section, we provide an overview of our classifer evaluation metrics. We use standard metrics to evaluate the performance of our classifier: *Precision*, *Recall* and *F1-Score*. We compute these metrics per class, where each class represents a webpage. We compute these metrics on a class as if it was a "one vs. all" binary classification: we call "positives" the samples that belong to that class and "negatives" the samples that belong to the rest of classes. Precision is the ratio of true positives to the total number of samples that were classified as positive (true positives and false positives). Recall is the ratio of true positives to the total number of positives (true positives and false negatives). The F1-score is the harmonic mean of precision and recall.

A.3 Estimation of Probabilities

In this section, we explain how we estimated the probabilities for the entropy analysis in Section 2.7.

We define the *anonymity set* of a trace s as a multiset:

$$A(s) := \{w^{\mathrm{m}_s(w)}\},$$

where $\mathrm{m}_s(w)$ is the multiplicity of a website w in $A(s)$. The multiplicity is a function defined as the number of times that trace s occurrs in w.

The probability $\Pr[W = w \mid S_l = o]$ can be worked out using Bayes. For instance, for website w,

$$\Pr[W = w \mid S_l = o] = \frac{\Pr[W{=}w]\,\Pr[S_l{=}o|W{=}w]}{\sum_{i=1}^{m} \Pr[W = w_i]\,\Pr[S_l = o \mid W = w_i]} \tag{A.1}$$

We assume the distribution of priors is uniform, i.e., the probability of observing a website is the same for all websites: $\Pr[w_i] = \Pr[w_j] \quad \forall i, j$.

We acknowledge that this is an unrealisitc assumption but we provide the mathematical model to incorporate the priors in case future work has the data to estimate them.

Assuming uniform priors simplifies the Bayes rule formula since we can factor out $\Pr[W = w_i]$ in Equation A.1

Regarding the likelihoods of observing the traces given a website, we can use the traffic trace samples in our dataset as observations to estimate them:

$$\Pr[S_l = o \mid W = w_i] \approx \frac{\mathrm{m}_s(w_i)}{k_i}.$$

Since we have a large number of samples for all the sites, we can fix the same sample size for all sites: $k_i = k_j \quad \forall i, j$. A fixed sample size allows us to factor out k_i in our likelihood estimates and, thus, the posterior can be estimated as:

$$\Pr[W = w \mid S_l = o] \approx \frac{\mathrm{m}_s(w)}{\sum_{i=1}^{m} \mathrm{m}_s(w_i)} = \frac{\mathrm{m}_s(w)}{|A(s)|}.$$

That is the multiplicity of website w divided by the size of the s's anonymity set, which can be computed efficiently for all w and s using vectorial operations.

A.4 Survivors and Easy Preys

In this section, we show results from our analysis of survivors and easy preys, as discussed in Section 2.5.3. We show the top-10 webpages with highest-mean and lowest-variance (Table A.2), lowest-mean and lowest-variance (Table A.3), and highest-variance F1-score (Table A.4).

A.5 Confusion Graphs

We have used *confusion graphs* to understand the errors of the classifier. Confusion graphs are the graph representation of confusion matrices. They allow to easily visualize large confusion matrices by representing misclassifications as directed graphs. Confusion graphs have been used in website fingerprinting [69] and other classification tasks to understand classifier error [268]. The graphs for our classifier can be found at https://github.com/spring-epfl/doh_traffic_analysis/blob/master/paper/doh_traffic_analysis_ndss2020.pdf.

Table A.2: Top-10 with highest-mean and lowest-variance F1-Score

Alexa Rank	Mean F1-Score	Stdev F1-Score	Domain name
777	0.95	0.08	militaryfamilygiftmarket.com
985	0.95	0.08	myffpc.com
874	0.95	0.08	montrealhealthygirl.com
712	0.95	0.08	mersea.restaurant
1496	0.95	0.08	samantha-wilson.com
1325	0.95	0.08	nadskofija-ljubljana.si
736	0.95	0.08	michaelnewnham.com
852	0.95	0.08	mollysatthemarket.net
758	0.95	0.08	midwestdiesel.com
1469	0.95	0.08	reclaimedbricktiles.blogspot.si

Table A.3: Top-10 sites with lowest-mean and lowest-variance F1-Score

Alexa Rank	Mean F1-Score	Stdev F1-Score	Domain name
822	0.11	0.10	mjtraders.com
1464	0.11	0.08	ravenfamily.org
853	0.14	0.09	moloneyhousedoolin.ie
978	0.14	0.17	mydeliverydoctor.com
999	0.17	0.10	myofascialrelease.com
826	0.17	0.11	mm-bbs.org
1128	0.17	0.10	inetgiant.com
889	0.18	0.14	motorize.com
791	0.18	0.15	mindshatter.com
1193	0.20	0.14	knjiznica-velenje.si

Table A.4: Top-10 sites with highest-variance F1-Score

Alexa Rank	Mean F1-Score	Stdev F1-Score	Domain name
1136	0.43	0.53	intothemysticseasons.tumblr.com
782	0.43	0.53	milliesdiner.com
766	0.43	0.53	mikaelson-imagines.tumblr.com
1151	0.43	0.53	japanese-porn-guidecom.tumblr.com
891	0.42	0.52	motorstylegarage.tumblr.com
909	0.42	0.52	mr-kyles-sluts.tumblr.com
918	0.44	0.52	mrsnatasharomanov.tumblr.com
1267	0.52	0.49	meander-the-world.com
238	0.48	0.49	caijing.com.cn
186	0.48	0.48	etsy.com

B Appendix for Chapter 3

B.1 Traceroute experiments at additional vantage points.

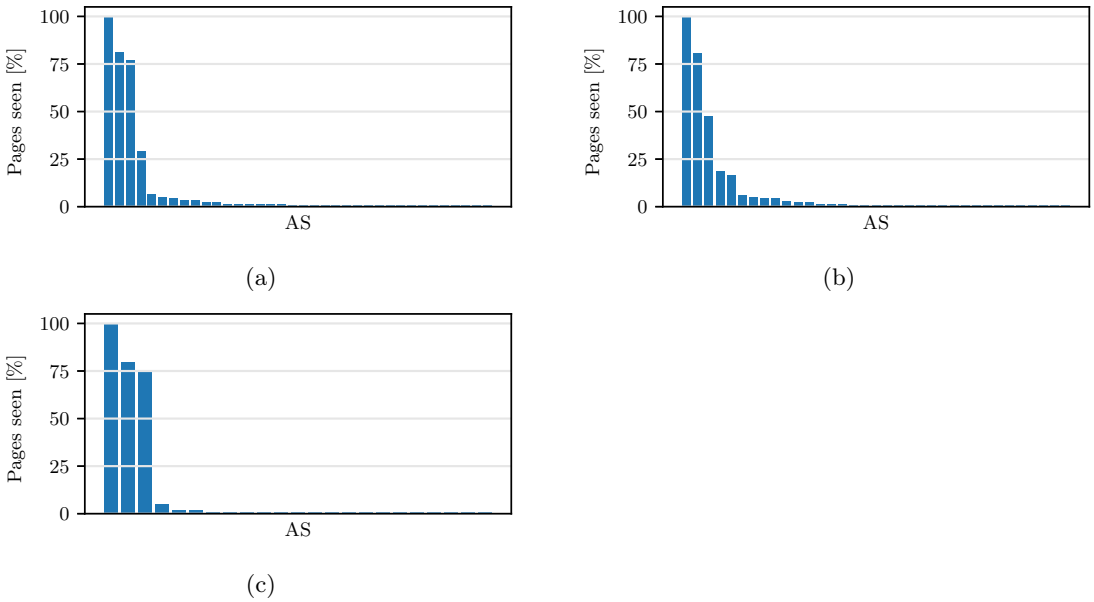


Figure B.1: Distribution of webpages seen by each AS, at three vantage points (a) Germany (b) United Kingdom (c) Singapore. Only the client's AS, Google, and Cloudflare observe >25% of the traffic.

The client location impacts the resources that might be fetched during a page load, and the paths taken by the network traffic to the destination servers. This, in turn, impacts the ASes that can view the traffic. In order to confirm that the trends we observe in our traffic visibility experiment (Section 3.4.2) hold at different locations, we collect

additional traceroutes from three additional vantage points located in Germany, UK, and Singapore.

Figure B.1 shows the distribution of webpages seen by different ASes, for our three vantage points. The number of total ASes we encounter on the traceroute are 36, 35, and 23. Out of these 13 ASes are common across all the vantage points. While the ASes that observe the traffic vary across locations, similar to Section 3.4.2 , only a small proportion of ASes that observe a large proportion of the traffic. Three ASes see more than 25% of the traffic for each vantage point: the client's AS, Google, and Cloudflare.

C Appendix for Chapter 4

C.1 Comparison between AdGraph and WebGraph features

Table C.1 compares the features used in WebGraph and AdGraph. WebGraph does not use content features. Graph size, Degree and Centrality features come under both structure and flow categories, since they include graph properties that are based on both normal and shared information edges. Some structural features used in AdGraph are not used in WebGraph due to WebGraph being adapted for offline use, whereas the features are useful in an online context.

C.2 Distribution of graph sizes

Figure C.1 shows the distribution of number of nodes in the graph representations of the web pages in our dataset. Since 80% of web pages have 250 nodes or fewer, we sample from this subset in our structural mutation experiments in Section 2.5.3.

C.3 Experimental run times

Figure C.2 describes the run times for the experiment described in Section 4.5.3 (adversary without collusion). Figure C.2(a) shows the impact of graph size on each iteration of the experiment. Smaller graphs have lower run times since features have to be calculated over a smaller number of nodes. Factors such as the complexity of the structure and flow behaviors also contribute towards time spent in each iteration, which explains variations in iteration time among graphs of the same size. We see that the mean time per iteration can be as high as ≈ 1200 seconds (median is ≈ 68 seconds). Figure C.2(b) shows the total

Table C.1: WebGraph features comparison with AdGraph. ● indicates that a feature is present. WebGraph calculates Graph size, Degree and Centrality features using both normal and shared information edges. The former comes under structural features while the latter comes under flow features.

Feature	Type	WebGraph	AdGraph
Request type (e.g. iframe, image)	Content		●
Ad keywords in request (e.g. banner, sponsor)	Content		●
Ad or screen dimensions in URL	Content		●
Valid query string parameters	Content		●
Length of URL	Content		●
Domain party	Content		●
Sub-domain check	Content		●
Base domain in query string	Content		●
Semi-colon in query string	Content		●
Graph size (# of nodes, # of edges, and nodes/edge ratio)	Structure	●	●
Degree (in, out, in+out, and average degree connectivity)	Structure	●	●
Centrality (closeness centrality, eccentricity)	Structure	●	
Number of siblings (node and parents)	Structure		●
Modifications by scripts (node and parents)	Structure		●
Parent's attributes	Structure		●
Parent degree (in, out, in+out, and average degree connectivity)	Structure		●
Sibling's attributes	Structure		●
Ascendant's attributes	Structure	●	●
Descendant of a script	Structure	●	●
Ascendant's script properties	Structure	●	●
Parent is an eval script	Structure	●	●
Local storage access (# of sets, # of gets)	Flow (storage)	●	
Cookie access (# of sets, # of gets)	Flow (storage)	●	
Requests (sent, received)	Flow (network)	●	
Redirects (sent, received, depth in chain)	Flow (network)	●	
Common access to the same storage node	Flow (shared information)	●	
Sharing of a storage node's value in a URL	Flow (shared information)	●	
Graph size (# of nodes, # of edges, and nodes/edge ratio)	Flow (shared information)	●	
Degree (in, out, in+out, and average degree connectivity)	Flow (shared information)	●	
Centrality (closeness centrality, eccentricity)	Flow (shared information)	●	

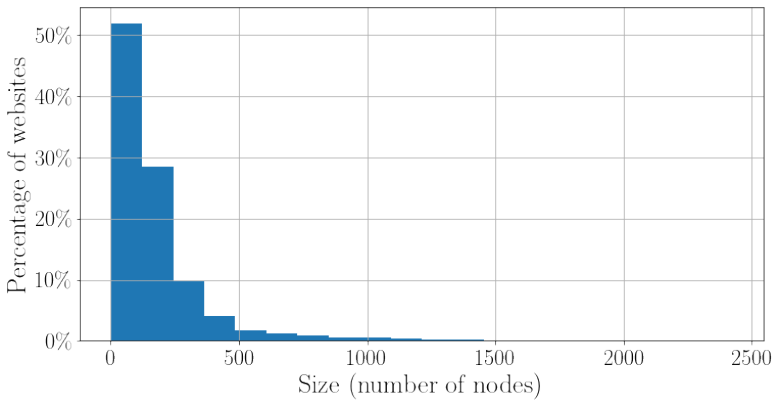
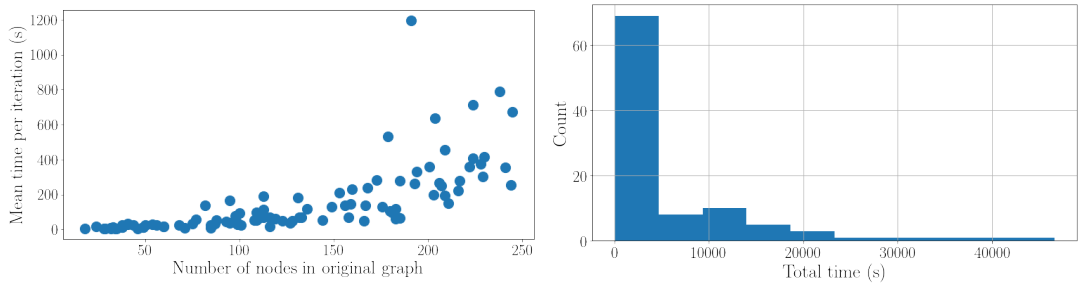


Figure C.1: Distribution of number of nodes in the graph representations of the web pages in the dataset.



(a) (b)

Figure C.2: Run-times for robustness experiments without collusion. Figure (a) shows mean time per iteration vs graph size, and (b) shows total run time.

experiment time over all iterations for a graph. Since we increase the sizes of graphs by 20% of their original size, bigger graphs will have a larger number of iterations. In our dataset, the maximum time taken for an experiment is 46654.19 seconds, the minimum is 15.67 seconds, and the median is 1745.11 seconds. 39% of the graphs in our dataset have a run time of more than an hour.

For the experiment in Section 4.5.3 (collusion with first party), the median time is 265.03 seconds, with the maximum time going up to 992.67 seconds, despite the maximum graph size being only 50 nodes. In comparison, for the adversary without collusion, for graph sizes up to 50 nodes, the median is 21.46 seconds and the maximum is 221.51 seconds. Since the adversary considers all nodes in the graph as potential parents, each iteration takes a longer amount of time.

C.4 Graph Mutation algorithm

In each iteration, the algorithm mutates WebGraph's graph representation and probes the model for classification decisions. The algorithm takes the following inputs: a graph representation of a web page, G_0, consisting of all the nodes in the graph; a set of nodes and edges T of size l_T, representing the resources loaded by the adversary, hereafter referred to as the adversary resources; a trained classifier M that identifies ATS in WebGraph; and a maximum number of iterations that the algorithm can run, `max_iter`.

The algorithm processes the input as follows: It first uses the classifier M to obtain classifications of all nodes in the original graph G_0 (lines 1–4 in Algorithm 1). Second, it iterates over the steps from lines 9–20 `max_iter` times. In each iteration, every adversary node tries resource addition, resource re-routing, and obfuscation, and produces a new mutated graph, G_i (line 11). Third, it extracts features from the mutated graph G_i and uses them to classify all the nodes in this graph (lines 11–12). Fourth, it compares

Algorithm 1 Greedy random graph mutation. G_0 is a web page representation, T is the set of l_T nodes and edges controlled by the adversary, M is a trained model, and `max_iter` is the maximum number of operations.

Input: $G_0, T, C, M,$ `max_iter`

for v $\in G_0$ **do**
  $\mathtt{x}_{G_0} \leftarrow \mathtt{ExtractFeatures}(v)\ \forall\ v \in G_0$
  $\mathtt{y}_{G_0} \leftarrow \mathtt{Classify}(M, \mathtt{x})\ \forall\ x\ in\ \mathtt{x}_{G_0}$
end for
$G \leftarrow G_0$
$i \leftarrow 0$
`graph-info` = []
while $i <$ `max_iter` **do**
  for $t \in T$ **do**
    $G_t \leftarrow \mathtt{MutateGraph}(G, t)$
    $\mathtt{x}_t \leftarrow \mathtt{ExtractFeatures}(v)\ \forall\ v \in G_t$
    $\mathtt{y}_t \leftarrow \mathtt{Classify}(M, \mathtt{x})\ \forall\ x\ in\ \mathtt{x}_t$
    $\mathtt{d}, \mathtt{u} \leftarrow \mathtt{GetDesiredAndUndesired}(\mathtt{y}_t, \mathtt{y}_{G_0})$
    $\Delta_t = \mathtt{d} - \mathtt{u}$
    `graph-info`$[t] \leftarrow (\Delta_t, t, G_t)$
  end for
  $G \leftarrow G_t$ in `graph-info`[t] with largest Δ_t
  $T \leftarrow \mathtt{UpdateAdv}(T, t \in \mathtt{graph\text{-}info}[t])$
  $T \leftarrow \mathtt{sample}(T, l_T)$
  $i \leftarrow i + 1$
end while

the predictions in the original and mutated graphs to obtain the number of desired and undesired switches (line 13). We assume an adversarial goal for which *desired* switches are all those in which an adversary node is switched from `ATS` to `Non-ATS`, whereas *undesired* switches are all those where any `Non-ATS` node is switched to `ATS` node. We call the total number of adversarial `ATS` nodes whose prediction the adversary wishes to change to `Non-ATS` the number of *required* switches. The switching of nodes not under the adversary's control from `ATS` to `Non-ATS` do not affect the adversary. These switches are, therefore, neither desired nor undesired. Finally, the adversary chooses the mutation that provides the best result, i.e., the one with the best trade-off between desired and undesired switches (lines 14–15). The adversary updates its T based on the chosen mutation (line 18). To keep memory and run time manageable, at the end of every iteration the algorithm randomly samples l_T adversarial nodes and edges from T (line 19) to be considered in the next iteration.

C.5 Mutations on a single web page.

To illustrate how mutations result in classification switches, we take as an example a web page in which the third party with the highest number of `ATS` resources is assets.wogaa.sg, which has 12 nodes in the graph. Figure C.3 shows the breakdown of classification switches as the adversary mutates the graph using the greedy mutation algorithm. The $\texttt{ATS}_{\texttt{Adv}}$ or the number of classifications the adversary wants to switch is 5 (pink line ——). From the adversary's point of view, adversarial nodes switching from `ATS` → `Non-ATS` are desired (blue line —●—), whereas adversarial nodes switching from `Non-ATS` → `ATS` are undesired (orange line —▲—). We consider `Non-ATS` → `ATS` changes on non-adversarial nodes to be undesired because they may have unintended impact on the web page (red line —◆— and brown line —■—). For instance, a first party `Non-ATS` → `ATS` switch may break the web page. We note that, if the adversary's goal is to just create a denial of service and force the user to disable ad and tracker blocking, the adversary might be unconcerned about breakage. In our experiments, switches that do not affect the adversary, such as `ATS` → `Non-ATS` for non-adversary nodes, are neither considered desired nor undesired (purple line —✶— and green line —●—).

There are two points worth highlighting from Figure C.3: (1) Even if an adversary achieves the maximum number of desired switches, the mutations may produce undesirable changes, to both the adversary's nodes and others. For instance, at 20% of growth, 3 of the adversary's `ATS` nodes are classified as `Non-ATS`, but also 7 `Non-ATS` nodes (3 adversary and 4 non-adversary) switch to the undesired `ATS` classification. (2) The evolution of desired and undesired switches is not monotonic, i.e. the classification of nodes may change in both directions as the adversary mutates the graph, resulting in increasing or decreasing counts. This finding reinforces our argument that it can be cumbersome for an adversary

to create targeted structural mutations without any unintended consequences. Not only it is hard to predict how mutations will affect adversary's own desired classification, but also how those mutations may result in undesirable changes to others.

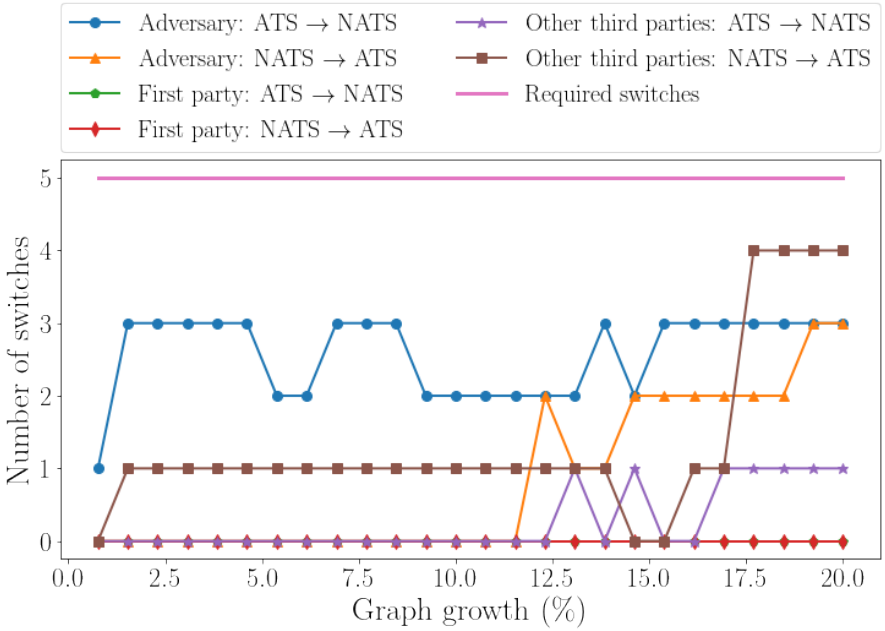


Figure C.3: Example breakdown of classification switches for the adversary's and other nodes on the graph. `NATS` is shorthand for `Non-ATS`. $\texttt{ATS}_{\texttt{Adv}} = 5$ (pink line), $\texttt{Non-ATS}_{\texttt{Adv}} = 7$, $\texttt{ATS}_{\texttt{Web}} = 62$, $\texttt{ATS}_{\texttt{Web}} = 13$ (not shown in plot). At 20% growth, the adversary achieves 3 `desired` switches, 7 `undesired` switches and 1 `neutral` switch. This leads to a success rate of 60%, a collateral damage of 10.14% and other changes of 7.7%.

C.6 WebGraph robustness experiments

We show plots for the experiments in Sections 4.5.3 and 4.5.3. Figure C.4 shows the results for an adversary that performs only resource addition against AdGraph (with only structural features) and WebGraph. AdGraph shows a higher number of successes for the adversary (44 pages with success rate $> 50\%$ as compared to 30 for WebGraph). At the same time, AdGraph also shows a higher amount of collateral damage (which is not beneficial for the adversary) – 66 pages with non-zero collateral damage, as compared to 47 for WebGraph. Hence, there is no clear-cut winner between the two classifiers in terms of robustness. However, we do see that AdGraph has lower successes and higher collateral damage than WebGraph against the powerful adversary that can do all mutations as shown in Figure 4.6 (note that this adversary cannot be used against AdGraph since AdGraph does not use information flow edges), since this adversary targets the effective, but costly, information sharing patterns.

Figure C.5 shows the results for an adversary that colludes against an adversary with no collusion (Section 4.5.3). A colluding adversary shows a higher number of successes (63 pages with success rate $> 50\%$ as compared to 60 for the non-colluding adversary), and a lower collateral damage (9 pages with damage $>0\%$ compared to 18 pages for a non-colluding adversary).

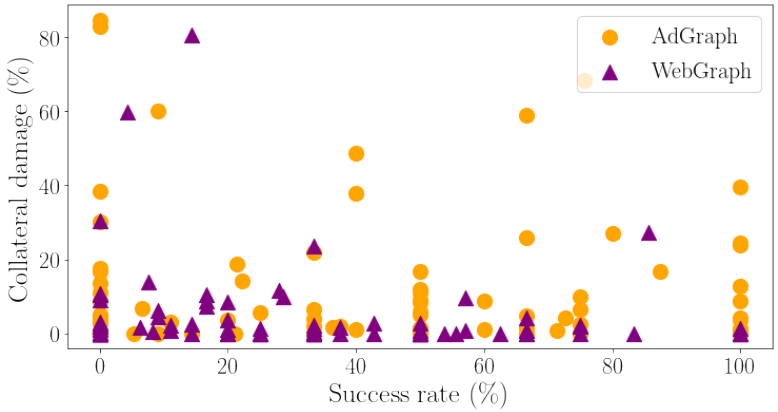


Figure C.4: Adversary's success rate vs. collateral damage for each test page at 20% graph growth, for resource addition against AdGraph and WebGraph.

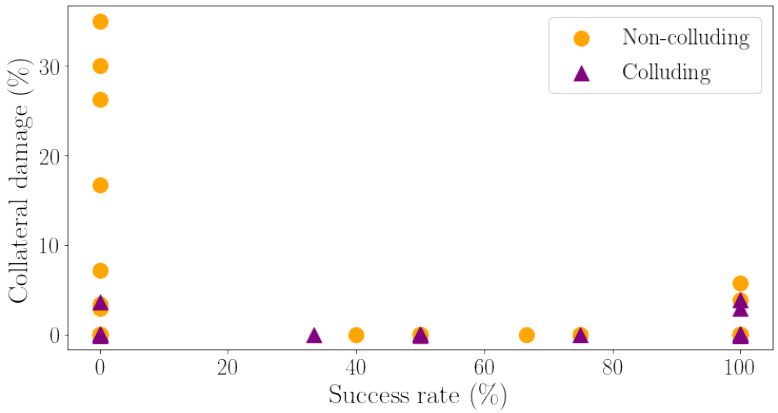


Figure C.5: Adversary's success rate vs. collateral damage for each test page at 20% graph growth, for colluding and non-colluding adversaries.

D Appendix for Chapter 5

D.1 Case Studies

In this section, we look at case studies of ATSes identified in Section 5.3.3 which are found to be extensively using first-party cookies for tracking purposes. We analyze the behavior of these ATS in our crawls, compare the observed behavior with their documentation, and create a generic model which all first-party-cookie-based ATSes follow in Section 5.3.4. We present case studies of three ATSes here: Lotame, ID5, and Criteo.

D.1.1 Lotame

Lotame is a data and identity management solution which claims to provide a single ID to users across multiple browsers, devices, and platforms. Lotame's Lightning Tag [238] packages the user visit data in a JSON object and sends it to its servers. Code D.1 shows an example payload sent to Lotame. The payload includes IDs assigned by the website, third-party identifiers present on the site, certain user behaviors (configured through collaboration between the publisher and Lotame), and other custom rules defined per website [269]. Lotame processes the payload and matches the data with its *Cartographer Identity Graph* [270], and sends back an ID, called `panoramaID` [271], which is stored as a first-party cookie or in localStorage.

D.1.2 ID5 Universal ID

ID5 provides identity resolution for publishers and advertisers through its *Identity Cloud* [240]. ID5's script packages a payload that contains several deterministic identifiers, such as email, usernames, and phone numbers (if available) and as well as probabilistic identifiers include, such as IP address, user agent, and location of the user [241]. ID5

then processes the payload and matches the data with its Identity Cloud and send back an ID, called *universal_id*, which is stored as a first-party cookie and as well as in local storage. An example payload from ID5 is shown in Figure D.2. We note that ID5 also provides *Partner Graph*, a service that enables information sharing among its partners [240]. Partner Graph allows different identify providers to exchange information with each other.

D.1.3 Criteo

Criteo provides *Criteo Identity Graph* for identity resolution [242]. Criteo Identity Graph is built from four different sources: (i) data contributed by advertisers, (ii) data collected from publisher websites by Criteo itself, (iii) data provided by Criteo partners such as LiveRamp and Oracle, (iv) and predictions on existing data by Criteo's machine learning models. Criteo claims that its identity graph is able to stitch together identifiers from more than 2 billion users across the world, and that it contains persistent deterministic identifiers for 96% of the users [242]. Similar to other identity resolution services, Criteo generates an ID, based on identifiers, such as hashed emails, mobile device IDs, cookie IDs, and stores it in first-party storage as `cto_bundle`. Their documentation shows that Criteo makes use of both first-party cookies and localStorage for storing `cto_bundle` cookie. We consider this to be one of the fundamental behaviors of first-party ATS cookies. As described in Section 5.4.1, WebGraph's graph representation abstracts storage to refer to both Cookies and localStorage. We also include a count of localStorage accesses in the feature set computed from the graph representation. Inclusion of these features help WebGraph effectively model first-party ATS cookies behavior.

```
data: {
  behaviorIds: [1,2,3],
  behaviors: {
    int: ['behaviorName', 'behaviorName2'],
    act: ['behaviorName']
  },
  ruleBuilder: {
    key1: ['value 1a', 'value 1b']
  },
  thirdParty: {
    namespace: 'NAMESPACE',
    value: 'TPID_VALUE'
  }
}
```

Code D.1: Example of data sent structure sent to Lotame during a user's first visit.

```
{
    "created_at":"2022-02-09T11:42:40.817811Z",
    "id5_consent":true,
    "original_uid":"ID5*FnFOGLkYzdJjuoK3KvAecVW2oFpZ7OrZiW7h-M0H
        ACAHYuWkxQrEGcpWuOkQUXbHB2OO8Rj0wt94jllT
        WHQ6wdkqOwSbnYea8cesuONCF4HZeIoDaB_TwBsy
        lKrs3tHB2Y87ZwPODrpYlGz1OG1Fgdn0YdgqoSGU
        SGxzS1gUzsHaMIUBVqf2IO8es6aUULEB2n48oyLO
        nGnRtstVqtcQQdquS3Aay4Hhgbzh9gIZyYHa_nLT
        d5rjbbROZXwkXDzB2yU1XUC2dukip1J_clVAgdtt
        xC_xaRRBOLiOfnvp9cHbqr_pWihTtaUMS_R6eLuB
        2_AMExt1UdhJZBe2mcXZAdwm9lcbeMMvlpg3MBrC
        oHcUgzNypi-5xLUqBD8GC4B3KsefcNkiDvI4n9ZL
        7OjAdzqB9PD-KczAx63Ck0gEIHdNPfQeEi-f5VaO
        OEhf6B3VUqDoL11hqVoIuDhKJbgd2kp0mgXabhwJ
        tPO7sgWwHd_vz_uIYYmqQBTbH-JFVB3h1-kI9GQv
        dby2PyftDawd596ho3tuOsKtoDOk4S4Her2Uw-_u
        BYRxrt6YzVYqB3tRwTVI3Fxm8cGJyjdmYAd88lom
        BIpkOeg2Ok4VTNc",
    "universal_uid":"ID5*HGH7W7iMpMu3-EPZCXUuqNBB7fFHUUVcbSddSSG
        Fu5UHYucsBxMz2jncvKS7rkwlB2MWiiPupapPxa
        79eieMAdkTyMQz82s1vIekPr28DEHZbqTCrapj9
        Fb9K0x4zjlB2YHOKNDwQY6mZwxk_1mwAdna3wWna
        hrpMEUrPxJSnAHaPYB-InS5DXGpQgqbqirB2nHFI
        D4j9i9BgCP3k0VygdqdtFHsT7eeDfFYuB8EQ0Ha4
        -yV9Ifvbvi5oxmtH7HB2xg-mmmOeyVOPBYGi2tfw
        dtREZnUE83cfn_LHvHvu4HbvkLkwEFJiddOEp4PT
        ZbB2-de_VPyKHax5JtpO46xwdwZ_0UMgANOsZygV
        0SrrMHcZ37qQB-LkCO4tWoTbv_B3KMGCMrebcfLE
        TeCn0AEgdzIR1utDJzM6AaiL9KVkAHdPtrAtTv73
        ZyDg92Rq-_B3XeRNOOc7b2CEBsilXOlQd2sfmR36
        NyW-dsK9CUmd4Hd3vcrlAWzfYEfw01Q5J1B3ibAF
        UYrA0XWMl-D9jSlAd5iX1tGA4vPu0wdZkXVOEHek
        q2xibOm9XwN2nSdZjbB3v8nOyzGuF9QgwI67pMGQ
        d85BszRCJDUkiiu-tv5BQ",
    "signature":"ID5_Ab6tnGgmCcjKo-qFGVKszuNpNePqkOHZT
        rbCmpuktLLOlNOCALhmY_91AHP8LUOBvfJT2Q
        JQWlsUEfynB1hBGZc",
    "link_type":1,
    "cascade_needed":true,
    "privacy":{
       "jurisdiction":"other",
       "id5_consent":true
    }
}
```


Code D.2: Example of data structure received from ID5 during a user's first visit.

D.2 Cross-Device User Attribution

The methodology for cross-site attribution can also be extended to cross-device attribution. In this slightly more complex scenario, the user is not only visiting different sites but also

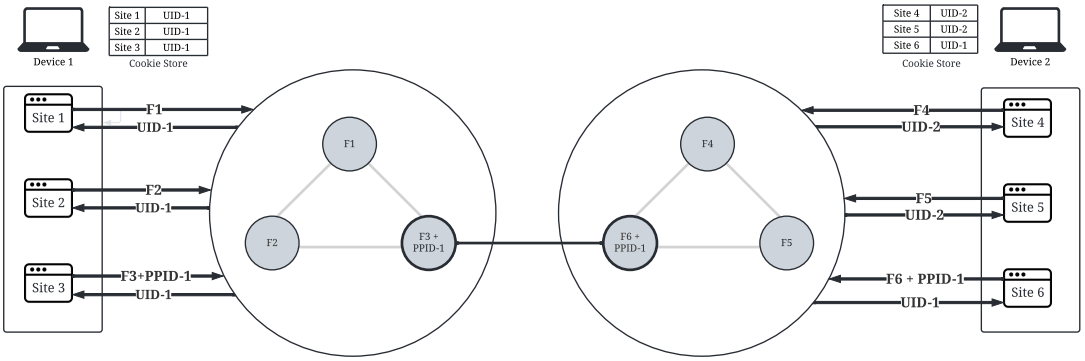


Figure D.1: This figure shows the flow of information and identifiers through an identity graph for a cross-device attribution. The user visits sites 1, 2, and 3 via Device 1. The identity graph returns a $UID-1$ for all the site visits, using a probabilistic matching of fingerprints $F1$, $F2$, and $F3$ sent on each respective website. A Publisher Provided ID $PPID-1$ is also sent alongside $F3$ when visiting site 3. The user visits sites 4, 5, and 6 through a new Device 2. Because the fingerprints $F4$ and $F5$ are different from $F1$, $F2$, and $F3$, the identity graph returns a new $UID-2$ for these site visits. On site 6, the website obtains and sends a Publisher Provided ID which matches $PPID-1$ provided on site 3. As a result, the identity graph matches and returns the existing user's $UID-1$ for site 6.

using different devices with different fingerprints. We show an example of cross-device attribution in Figure D.1. Instead of visiting the sites on the same device, the user now visits sites 1-3 on device 1, and then visits sites 4-6 on device 2. Fingerprints F_1, F_2 and F_3 are collected on sites 1-3 while using device 1, and F_4, F_5 and F_6 are collected while using device 2. Sites 3 and 6 ask the user for an additional $PPID-1$. There is no similarity in fingerprints between the two devices. However, as sites 3 and 6 collect the additional $PPID-1$, the tracker is able to identify and populate its first-party cookie on each of those sites with the correct user ID (UID_1).

Bibliography

[1] Reuben Binns et al. Tracking on the web, mobile and the internet of things. *Foundations and Trends® in Web Science*, 8(1–2):1–113, 2022.

[2] Christian Grothoff, Matthias Wachs, Monika Ermert, and Jacob Appelbaum. NSA's MORECOWBELL: Knell for DNS. Unpublished technical report, 2017.

[3] The NSA files decoded. https://www.theguardian.com/us-news/the-nsa-files. Accessed: 2019-05-13.

[4] Earl Zmijewski. Turkish Internet Censorship Takes a New Turn. https://dyn.com/blog/turkish-internet-censorship, 2014. Accessed: 2018-12-06.

[5] Nicholas Weaver, Christian Kreibich, and Vern Paxson. Redirecting DNS for Ads and Profit. In *FOCI*, 2011.

[6] S. Bortzmeyer. DNS Privacy Considerations. Internet Requests for Comments, 2015.

[7] A.Cooper, H. Tschofenig, B. Aboba, J. Peterson, J. Morris, M. Hansen, and R. Smith. Privacy Considerations for Internet Protocols. Internet Requests for Comments, 2015.

[8] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman. Specification for DNS over Transport Layer Security (TLS). RFC 7858, RFC Editor, May 2016.

[9] P. Hoffman and P. McManus. DNS Queries over HTTPS (DoH). RFC 8484, RFC Editor, October 2018.

[10] RFC 9000. https://datatracker.ietf.org/doc/html/rfc9000. Accessed: 2021-08-12.

[11] Google Public DNS over HTTPS (DoH) supports RFC 8484 standard. https://security.googleblog.com/2019/06/google-public-dns-over-https-doh.html. Accessed: 2019-04-03.

[12] Cloudflare DNS over HTTPS. https://developers.cloudflare.com/1.1.1.1/dns-over-https/. Accessed: 2018-05-07.

[13] Mozilla Support. Firefox DNS-over-HTTPS: About the US rollout of DNS over HTTPS. https://support.mozilla.org/en-US/kb/firefox-dns-over-https#w_about-the-us-rollout-of-dns-over-https. Accessed: 2020-01-07.

[14] Jean-Pierre Smith, Prateek Mittal, and Adrian Perrig. Website Fingerprinting in the Age of QUIC. *PETS*, 2021(2):48–69, 2021.

[15] RFC 9000, Section 19.1 PADDING Frames. https://datatracker.ietf.org/doc/html/rfc9000#section-19.1. Accessed: 2021-08-12.

[16] A. Mayrhofer. The EDNS(0) Padding Option. RFC 7830, RFC Editor, May 2016.

[17] Selena Deckelmann. DNS over HTTPS (DoH) – testing on beta. https://blog.mozilla.org/futurereleases/2018/09/13/dns-over-https-doh-testing-on-beta, 2018. accessed: 2018-12-30.

[18] Geoff Huston. DOH! DNS over HTTPS explained. https://labs.ripe.net/Members/gih/doh-dns-over-https-explained, 2018. Accessed: 2018-12-27.

[19] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. Website Fingerprinting at Internet Scale. In *NDSS*, 2016.

[20] Tao Wang and Ian Goldberg. On Realistically Attacking Tor with Website Fingerprinting. *Proc. Priv. Enhancing Technol.*, 2016(4):21–36, 2016.

[21] Jamie Hayes and George Danezis. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1187–1203, Austin, TX, August 2016. USENIX Association.

[22] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1928–1943, 2018.

[23] Andrew M. White, Austin R. Matthews, Kevin Z. Snow, and Fabian Monrose. Phonotactic Reconstruction of Encrypted VoIP Conversations: Hookt on Fon-iks. In *IEEE Symposium on Security and Privacy (S&P 2011)*, 2011.

[24] Se Eun Oh, Shuai Li, and Nicholas Hopper. Fingerprinting Keywords in Search Queries over Tor. *PoPETs*, 2017.

[25] The DNS Privacy Project. Initial Performance Measurements (Q1 2018). https://dnsprivacy.org/wiki/pages/viewpage.action?pageId=14025132, 2018. Accessed: 2018-12-27.

[26] The DNS Privacy Project. Initial Performance Measurements (Q4 2018). https://dnsprivacy.org/wiki/pages/viewpage.action?pageId=17629326, 2018. Accessed: 2018-12-27.

[27] Austin Hounsel, Kevin Borgolte, Paul Schmitt, Jordan Holland, and Nick Feamster. Analyzing the Costs (and Benefits) of DNS, DoT, and DoH for the Modern Web. *CoRR*, abs/1907.08089, 2019.

[28] Timm Bottger, Felix Cuadrado, Gianni Antichi, Eder Leao Fernandes, Gareth Tyson, Ignacio Castro, and Steve Uhlig. An Empirical Study of the Cost of DNS-over-HTTPs. In *Internet Measurement Conference (IMC)*, 2019.

[29] Brad Miller, Ling Huang, Anthony D Joseph, and J Doug Tygar. I know why you went to the clinic: Risks and realization of HTTPS traffic analysis. In *Privacy Enhancing Technologies Symposium (PETS)*, pages 143–163. Springer, 2014.

[30] Xiapu Luo, Peng Zhou, Edmond WW Chan, Wenke Lee, Rocky KC Chang, Roberto Perdisci, et al. HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. In *NDSS*, 2011.

[31] Milad Nasr, Amir Houmansadr, and Arya Mazumdar. Compressive traffic analysis: A new paradigm for scalable traffic analysis. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2053–2069, 2017.

[32] DNS over Tor. https://developers.cloudflare.com/1.1.1.1/fun-stuff/dns-over-tor/. Accessed: 2018-12-09.

[33] Tao Wang and Ian Goldberg. Improved Website Fingerprinting on Tor. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 201–212. ACM, 2013.

[34] iodine. https://code.kryo.se/iodine/. Accessed: 2019-05-13.

[35] Spamhaus. https://www.spamhaus.org/zen/. Accessed: 2019-05-13.

[36] Anonymous. Towards a comprehensive picture of the Great Firewall's DNS censorship. In *USENIX Security Symposium*. USENIX Association, 2014.

[37] Paul Pearce, Ben Jones, Frank Li, Roya Ensafi, Nick Feamster, Nick Weaver, and Vern Paxson. Global measurement of DNS manipulation. In *USENIX Security Symposium. USENIX*, page 22, 2017.

[38] Saikat Guha and Paul Francis. Identity Trail: Covert Surveillance Using DNS. In *Privacy Enhancing Technologies Symposium (PETS)*, 2007.

[39] DNSSEC: DNS Security Extensions. https://www.dnssec.net/. Accessed: 2018-12-09.

[40] DNSCrypt. https://dnscrypt.info/. Accessed: 2018-12-09.

[41] Kenji Baheux. Experimenting with same-provider DNS-over-HTTPS upgrade. https://blog.mozilla.org/futurereleases/2018/09/13/dns-over-https-doh-testing-on-beta, 2019. accessed: 2019-09-13.

[42] T. Reddy, D. Wing, and P. Patil. DNS over Datagram Transport Layer Security (DTLS). RFC 8094, RFC Editor, February 2017.

[43] Specification of DNS over Dedicated QUIC Connections. https://www.ietf.org/id/draft-huitema-quic-dnsoquic-05.txt. Accessed: 2018-12-09.

[44] Haya Shulman. Pretty bad privacy: Pitfalls of DNS encryption. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. ACM, 2014.

[45] Dominik Herrmann, Christian Banse, and Hannes Federrath. Behavior-based tracking: Exploiting characteristic patterns in DNS traffic. *Computers & Security*, 2013.

[46] Basileal Imana, Aleksandra Korolova, and John Heidemann. Enumerating privacy leaks in DNS data collected above the recursive. In *NDSS: DNS Privacy Workshop*, 2018.

[47] DNS-OARC: Domain Name System Operations Analysis and Research Center. https://www.dns-oarc.net/tools/dsc. Accessed: 2018-11-26.

[48] DNS-STATS: ICANN's IMRS DNS Statistics. https://www.dns.icann.org/imrs/stats. Accessed: 2018-12-06.

[49] Use DNS data to identify malware patient zero. https://docs.splunk.com/Documentation/ES/5.2.0/Usecases/PatientZero. Accessed: 2018-12-06.

[50] DNS Analytics. https://constellix.com/dns/dns-analytics/. Accessed: 2018-12-06.

[51] Anonymous. The Collateral Damage of Internet Censorship by DNS Injection. *SIGCOMM Comput. Commun. Rev.*, 42(3):21–27, 2012.

[52] Alerts about BGP hijacks, leaks, and outages. https://bgpstream.com/. Accessed: 2019-05-13.

[53] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *ACM Conference on Computer and Communications Security (CCS)*, pages 605–616. ACM, 2012.

[54] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *USENIX Security Symposium*, pages 143–157. USENIX Association, 2014.

[55] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated Website Fingerprinting through Deep Learning. In *NDSS*, 2018.

[56] Allison McDonald, Matthew Bernhard, Luke Valenta, Benjamin VanderSloot, Will Scott, Nick Sullivan, J Alex Halderman, and Roya Ensafi. 403 Forbidden: A Global View of CDN Geoblocking. In *Proceedings of the Internet Measurement Conference 2018*, pages 218–230. ACM, 2018.

[57] Roberto Gonzalez, Claudio Soriente, and Nikolaos Laoutaris. User Profiling in the Time of HTTPS. In *Proceedings of the 2016 Internet Measurement Conference*, pages 373–379. ACM, 2016.

[58] Marc Liberatore and Brian Neil Levine. "Inferring the source of encrypted HTTP connections". In *ACM Conference on Computer and Communications Security (CCS)*, pages 255–263. ACM, 2006.

[59] Kevin P Dyer, Scott E Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *2012 IEEE symposium on security and privacy*, pages 332–346. IEEE, 2012.

[60] Qixiang Sun, Daniel R Simon, Yi-Min Wang, Wilf Russel, Venkata N. Padmanabhan, and Lili Qiu. Statistical identification of encrypted web browsing traffic. In *IEEE Symposium on Security and Privacy (S&P)*, pages 19–30. IEEE, 2002.

[61] Andrew Hintz. Fingerprinting websites using traffic analysis. In *Privacy Enhancing Technologies Symposium (PETS)*, pages 171–178. Springer, 2003.

[62] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial Naïve-Bayes classifier. In *ACM Workshop on Cloud Computing Security*, pages 31–42. ACM, 2009.

[63] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 103–114. ACM, 2011.

[64] Mario Almeida, Alessandro Finamore, Diego Perino, Narseo Vallina-Rodriguez, and Matteo Varvello. Dissecting DNS Stakeholders in Mobile Networks. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 28–34. ACM, 2017.

[65] David Plonka and Paul Barford. Context-aware clustering of DNS query traffic. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pages 217–230. ACM, 2008.

[66] Rebekah Houser, Zhou Li, Chase Cotton, and Haining Wang. An investigation on information leakage of DNS over TLS. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies (CONEXT)*, pages 123–137, 2019.

[67] Rob Jansen, Marc Juarez, Rafael Galvez, Tariq Elahi, and Claudia Diaz. Inside Job: Applying Traffic Analysis to Measure Tor from Within. In *Network & Distributed System Security Symposium (NDSS)*. Internet Society, 2018.

[68] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[69] Rebekah Overdorf, Marc Juarez, Gunes Acar, Rachel Greenstadt, and Claudia Diaz. How unique is your onion? An Analysis of the Fingerprintability of Tor Onion Services. In *ACM Conference on Computer and Communications Security (CCS)*, pages 2021–2036. ACM, 2017.

[70] Marc Juarez, Mohsen Imani, Mike Perry, Claudia Diaz, and Matthew Wright. Toward an efficient website fingerprinting defense. In *European Symposium on Research in Computer Security*, pages 27–46. Springer, 2016.

[71] Ariel Stolerman, Rebekah Overdorf, Sadia Afroz, and Rachel Greenstadt. Breaking the Closed-World Assumption in Stylometric Authorship Attribution. In *IFIP Int. Conf. Digital Forensics*, 2014.

[72] Muhammad Ahmad Bashir and Christo Wilson. Diffusion of user tracking data in the online advertising ecosystem. *Privacy Enhancing Technologies Symposium (PETS)*, 2018.

[73] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. A critical evaluation of website fingerprinting attacks. In *ACM Conference on Computer and Communications Security (CCS)*, pages 263–274. ACM, 2014.

[74] John S Otto, Mario A Sánchez, John P Rula, and Fabián E Bustamante. Content delivery and the natural evolution of DNS: remote DNS trends, performance issues and alternative solutions. In *Proceedings of the 2012 Internet Measurement Conference*, pages 523–536. ACM, 2012.

[75] John P Rula and Fabian E Bustamante. Behind the curtain: Cellular DNS and content replica selection. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 59–72. ACM, 2014.

[76] Platon Kotzias, Abbas Razaghpanah, Johanna Amann, Kenneth G Paterson, Narseo Vallina-Rodriguez, and Juan Caballero. Coming of Age: A Longitudinal Study of TLS Deployment. In *Proceedings of the Internet Measurement Conference*, pages 415–428. ACM, 2018.

[77] J. Damas, M. Graff, and P. Vixie. Extension Mechanisms for DNS (EDNS(0)). RFC 6891, RFC Editor, April 2013.

[78] Padding Policies for Extension Mechanisms for DNS (EDNS(0)). https://tools.ietf.org/html/rfc8467. Accessed: 2019-05-10.

[79] Michael Carl Tschantz, Sadia Afroz, Anonymous, and Vern Paxson. SoK: Towards grounding censorship circumvention in empiricism. In *IEEE Symposium on Security and Privacy (S&P)*, pages 914–933. IEEE, 2016.

[80] Sheharbano Khattak, Tariq Elahi, Laurent Simon, Colleen M Swanson, Steven J Murdoch, and Ian Goldberg. SoK: Making sense of censorship resistance systems. *Privacy Enhancing Technologies Symposium (PETS)*, 2016(4):37–61, 2016.

[81] URL testing lists intended for discovering website censorship. https://github.com/citizenlab/test-lists. Accessed: 2019-09-11.

[82] Paul Schmitt, Anne Edmundson, Allison Mankin, and Nick Feamster. Oblivious DNS: practical privacy for DNS queries. *Proceedings on Privacy Enhancing Technologies*, 2019(2):228–244, 2019.

[83] TLS Encrypted Client Hello. https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-12. Accessed: 2021-07-05.

[84] Sudheesh Singanamalla, Suphanat Chunhapanya, Marek Vavruša, Tanya Verma, Peter Wu, Marwan Fayed, Kurtis Heimerl, Nick Sullivan, and Christopher Wood. Oblivious DNS over HTTPS (ODoH): A Practical Privacy Enhancement to DNS. *PETS*, 2021.

[85] Heyning Cheng and Ron Avnur. Traffic analysis of SSL encrypted web browsing. *Project paper, University of Berkeley*, 1998. Available at http://www.cs.berkeley.edu/~daw/teaching/cs261-f98/projects/final-reports/ronathan-heyning.ps.

[86] Sandra Siby, Marc Juarez, Claudia Diaz, Narseo Vallina-Rodriguez, and Carmela Troncoso. Encrypted DNS–> Privacy? A traffic analysis perspective. In *NDSS*, 2020.

[87] Usage statistics of HTTP/3 for websites. https://w3techs.com/technologies/details/ce-http3. Accessed: 2022-05-27.

[88] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. Var-CNN: A data-efficient website fingerprinting attack based on deep learning. *PETS*, 2019.

[89] Payap Sirinam, Nate Mathews, Mohammad Saidur Rahman, and Matthew Wright. Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1131–1148, 2019.

[90] Xiang Cai, Rishab Nithyanand, and Rob Johnson. CS-BuFLO: A congestion sensitive website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 121–130, 2014.

[91] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 227–238, 2014.

[92] Jiajun Gong and Tao Wang. Zero-delay lightweight defenses against website fingerprinting. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 717–734, 2020.

[93] Shawn Shan, Arjun Nitin Bhagoji, Haitao Zheng, and Ben Y Zhao. A Real-time Defense against Website Fingerprinting Attacks. In *ACM Workshop on Artificial Intelligence and Security (AISec'21)*, 2021.

[94] Mohammad Saidur Rahman, Mohsen Imani, Nate Mathews, and Matthew Wright. Mockingbird: Defending against deep-learning-based website fingerprinting attacks with adversarial traces. *IEEE Transactions on Information Forensics and Security*, 16:1594–1609, 2020.

[95] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. Defeating DNN-Based Traffic Analysis Systems in Real-Time With Blind Adversarial Perturbations. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[96] BLANKET. https://github.com/SPIN-UMass/BLANKET. Accessed: 2022-05-25.

[97] Mike Perry. Experimental Defense for Website Traffic Fingerprinting. https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting. Accessed: 2021-10-03.

[98] Giovanni Cherubin, Jamie Hayes, and Marc Juarez. "Website fingerprinting defenses at the application layer". In *Privacy Enhancing Technologies Symposium (PETS)*, pages 168–185. De Gruyter, 2017.

[99] Nick Feamster and Roger Dingledine. Location diversity in anonymity networks. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 66–76, 2004.

[100] Steven J Murdoch and Piotr Zieliński. Sampled traffic analysis by internet-exchange-level adversaries. In *International workshop on privacy enhancing technologies*, pages 167–183. Springer, 2007.

[101] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users get routed: Traffic correlation on Tor by realistic adversaries. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 337–348, 2013.

[102] Simran Patil and Nikita Borisov. What can you learn from an IP? In *Proceedings of the Applied Networking Research Workshop*, pages 45–51, 2019.

[103] Alexa 1M. http://s3.amazonaws.com/alexa-static/top-1m.csv.zip. Accessed: 2021-07-05.

[104] Cisco Umbrella Top 1M Domains List. https://www.trisul.org/devzone/doku.php/cisco_umbrella_top-1m_domains_list. Accessed: 2021-07-05.

[105] The Majestic Million. https://majestic.com/reports/majestic-million. Accessed: 2021-07-05.

[106] Jiasong Bai, Menghao Zhang, Guanyu Li, Chang Liu, Mingwei Xu, and Hongxin Hu. FastFE: Accelerating ML-based traffic analysis with programmable switches. In *Proceedings of the Workshop on Secure Programmable Network Infrastructure*, pages 1–7, 2020.

[107] Diogo Barradas, Nuno Santos, Luis Rodrigues, Salvatore Signorello, Fernando MV Ramos, and André Madeira. FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications. In *Proceedings of the 28th Network and Distributed System Security Symposium (San Diego, CA, USA*, 2021.

[108] Davide Tammaro, Silvio Valenti, Dario Rossi, and Antonio Pescapé. Exploiting packet-sampling measurements for traffic characterization and classification. *International Journal of Network Management*, 22(6):451–476, 2012.

[109] Valentín Carela-Español, Pere Barlet-Ros, Albert Cabellos-Aparicio, and Josep Solé-Pareta. Analysis of the impact of sampling on NetFlow traffic classification. *Computer Networks*, 55(5):1083–1099, 2011.

[110] Joshua Juen, Aaron Johnson, Anupam Das, Nikita Borisov, and Matthew Caesar. Defending Tor from Network Adversaries: A Case Study of Network Path Prediction. *Proceedings on Privacy Enhancing Technologies*, 2015(2):171–187, 2015.

[111] George Nomikos and Xenofontas Dimitropoulos. traIXroute: Detecting IXPs in traceroute paths. In *International Conference on Passive and Active Network Measurement*, pages 346–358. Springer, 2016.

[112] João Marco C Silva, Paulo Carvalho, and Solange Rito Lima. Inside packet sampling techniques: exploring modularity to enhance network measurements. *International Journal of Communication Systems*, 30(6):e3135, 2017.

[113] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. Sketch-based change detection: Methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 234–247, 2003.

[114] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114, 2016.

[115] Paul Tune and Darryl Veitch. OFSS: Skampling for the flow size distribution. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 235–240, 2014.

[116] DuckDuckGo Tracker Radar. https://github.com/duckduckgo/tracker-radar. Accessed: 2021-07-05.

[117] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *CCS*, 2016.

[118] Emily Hentgen. Measuring the Security of Website Fingerprinting Defenses. https://infoscience.epfl.ch/record/289258?&ln=en, 2019. Accessed: 2021-10-08.

[119] Web Bundles. https://wicg.github.io/webpackage/draft-yasskin-wpack-bundled-exchanges.html. Accessed: 2021-10-08.

[120] Yashodhar Govil, Liang Wang, and Jennifer Rexford. MIMIQ: Masking IPs with Migration in QUIC. In *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*, 2020.

[121] Near-path NAT for IP Privacy. https://github.com/bslassey/ip-blindness/blob/master/near_path_nat.md. Accessed: 2021-10-08.

[122] Multiplexed Application Substrate over QUIC Encryption (MASQUE). https://datatracker.ietf.org/wg/masque/about/. Accessed: 2021-10-08.

[123] Mona Wang, Anunay Kulshrestha, Liang Wang, and Prateek Mittal. Leveraging strategic connection migration-powered traffic splitting for privacy. *Proceedings on Privacy Enhancing Technologies*, 1:18, 2022.

[124] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[125] uBlock Origin. https://github.com/gorhill/uBlock.

[126] Ghostery. https://www.ghostery.com.

[127] MDN. Storage access policy: Block cookies from trackers. https://developer.mozilla.org/en-US/docs/Mozilla/Firefox/Privacy/Storage_access_policy.

[128] MDN. Redirect tracking protection. https://developer.mozilla.org/en-US/docs/Mozilla/Firefox/Privacy/Redirect_Tracking_Protection.

[129] Microsoft Edge Team. Introducing tracking prevention, now available in Microsoft Edge preview builds. https://blogs.windows.com/msedgedev/2019/06/27/tracking-prevention-microsoft-edge-preview/.

[130] Brave. A Long List of Ways Brave Goes Beyond Other Browsers to Protect Your Privacy. https://brave.com/privacy-features/.

[131] EasyList. https://easylist.to/easylist/easylist.txt.

[132] EasyPrivacy. https://easylist.to/easylist/easyprivacy.txt.

[133] Disconnect. https://disconnect.me/.

[134] Sruti Bhagavatula, Christopher Dunn, Chris Kanich, Minaxi Gupta, and Brian Ziebart. Leveraging Machine Learning to Improve Unwanted Resource Filtering. In *WAIS*, 2014.

[135] David Gugelmann, Markus Happe, Bernhard Ager, and Vincent Lenders. An Automated Approach for Complementing Ad Blockers' Blacklists. In *PETS*, 2015.

[136] Anastasia Shuba, Athina Markopoulou, and Zubair Shafiq. NoMoAds: Effective and Efficient Cross-App Mobile Ad-Blocking. In *PETS*, 2018.

[137] Qianru Wu, Qixu Liu, Yuqing Zhang, Peng Liu, and Guanxing Wen. A Machine Learning Approach for Detecting Third-Party Trackers on the Web. In *ESORICS*, 2016.

[138] Andrew J. Kaizer and Minaxi Gupta. Towards Automatic identification of JavaScript-oriented Machine-Based Tracking. In *IWSPA*, 2016.

[139] 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 . In *PETS*, 2017.

[140] Peter Snyder, Antoine Vastel, and Benjamin Livshits. Who Filters the Filters: Understanding the Growth, Usefulness and Efficiency of Crowdsourced Ad Blocking. In *SIGMETRICS*, 2020.

[141] Philippe Skolka, Cristian-Alexandru Staicu, and Michael Pradel. Anything to hide? Studying minified and obfuscated code in the web. In *WWW*, pages 1735–1746, 2019.

[142] Umar Iqbal, Peter Snyder, Shitong Zhu, Benjamin Livshits, Zhiyun Qian, and Zubair Shafiq. AdGraph: A Graph-Based Approach to Ad and Tracker Blocking. In *S&P*, 2020.

[143] Alexander Sjösten, Peter Snyder, Antonio Pastor, Panagiotis Papadopoulos, and Benjamin Livshits. Filter List Generation for Underserved Regions. In *WWW*, 2020.

[144] Ha Dao, Johan Mazel, and Kensuke Fukuda. Characterizing CNAME Cloaking-Based Tracking on the Web. In *TMA*, 2020.

[145] Yana Dimova, Gunes Acar, Lukasz Olejnik, Wouter Joosen, and Tom Van Goethem. The CNAME of the Game: Large-scale Analysis of DNS-based Tracking Evasion. *PETS*, 2021.

[146] Panagiotis Papadopoulos, Nicolas Kourtellis, and Evangelos P. Markatos. Cookie Synchronization: Everything You Always Wanted to Know But Were Afraid to Ask. In *WWW*, 2019.

[147] Hieu Le, Athina Markopoulou, and Zubair Shafiq. CV-INSPECTOR: Towards Automating Detection of Adblock Circumvention. In *NDSS*, 2021.

[148] Imane Fouad, Nataliia Bielova, Arnaud Legout, and Natasa Sarafijanovic-Djukic. Missed by Filter Lists: Detecting Unknown Third-Party Trackers with Invisible Pixels. *PETS*, 2020.

[149] John Cook, Rishab Nithyanand, and Zubair Shafiq. Inferring tracker-advertiser relationships in the online advertising ecosystem using header bidding. *arXiv preprint arXiv:1907.07275*, 2019.

[150] Adblock Plus. https://adblockplus.org/.

[151] Umar Iqbal, Zubair Shafiq, and Zhiyun Qian. The Ad Wars: Retrospective Measurement and Analysis of Anti-Adblock Filter Lists. In *IMC*, 2017.

[152] Catalin Cimpanu. Ad Network Uses DGA Algorithm to Bypass Ad Blockers and Deploy In-Browser Miners. https://www.bleepingcomputer.com/news/security/ad-network-uses-dga-algorithm-to-bypass-ad-blockers-and-deploy-in-browser-miners/, 2018.

[153] Zhang Zaifeng. Who is Stealing My Power III: An Adnetwork Company Case Study, 2018. http://blog.netlab.360.com/who-is-stealing-my-power-iii-an-adnetwork-company-case-study-en/.

[154] Andrew Bosworth. A New Way to Control the Ads You See on Facebook, and an Update on Ad Blocking. https://newsroom.fb.com/news/2016/08/a-new-way-to-control-the-ads-you-see-on-facebook-and-an-update-on-ad-blocking/, 2016.

[155] Garett Sloane. Ad Blocker's Successful Assault on Facebook Enters Its Second Month. http://adage.com/article/digital/blockrace-adblock/311103/, 2017.

[156] Ben Williams. Ping pong with Facebook. https://adblockplus.org/blog/ping-pong-with-facebook, 2018.

[157] Weihang Wang, Yunhui Zheng, Xinyu Xing, Yonghwi Kwon, Xiangyu Zhang, and Patrick Eugster. WebRanz: Web Page Randomization For Better Advertisement Delivery and Web-Bot Prevention. In *FSE*, 2016.

[158] Mshabab Alrizah, Sencun Zhu, Xinyu Xing, and Gang Wang. Errors, Misunderstandings, and Attacks: Analyzing the Crowdsourcing Process of Ad-blocking Systems. In *IMC*, 2019.

[159] Hung Dang, Yue Huang, and Eechien Chang. Evading Classifiers by Morphing in the Dark. In *CCS*, 2017.

[160] Aurore Fass, Michael Backes, and Ben Stock. HideNoSeek: Camouflaging Malicious JavaScript in Benign ASTs. In *CCS*, 2019.

[161] Niels Hansen, Lorenzo De Carli, and Drew Davidson. Assessing Adaptive Attacks Against Trained JavaScript Classifiers. In *SecureComm*, 2020.

[162] Abdul Haddi Amjad, Danial Saleem, Muhammad Ali Gulzar, Zubair Shafiq, and Fareed Zaffar. Trackersift: Untangling mixed tracking and functional web resources. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 569–576, 2021.

[163] Quan Chen, Peter Snyder, Ben Livshits, and Alexandros Kapravelos. Detecting Filter List Evasion With Event-Loop-Turn Granularity JavaScript Signatures. In *S&P*, 2021.

[164] Amir Hossein Kargaran, Mohammad Sadegh Akhondzadeh, Mohammad Reza Heidarpour, Mohammad Hossein Manshaei, Kave Salamatian, and Masoud Nejad Sattary. On Detecting Hidden Third-Party Web Trackers with a Wide Dependency Chain Graph: A Representation Learning Approach. *arXiv preprint arXiv:2004.14826*, 2020.

[165] Shitong Zhu, Zhongjie Wang, Xun Chen, Shasha Li, Umar Iqbal, Zhiyun Qian, Kevin S Chan, Srikanth V Krishnamurthy, and Zubair Shafiq. A4: Evading Learning-based Adblockers. *arXiv preprint arXiv:2001.10999*, 2020.

[166] Florian Tramèr, Pascal Dupré, Gili Rusak, Giancarlo Pellegrino, and Dan Boneh. AdVersarial: Perceptual Ad Blocking Meets Adversarial Machine Learning. In *CCS*, 2019.

[167] Bypassing ad blockers for Google Analytics. https://analytics-bypassing-adblockers.netlify.app/.

[168] Bjørn Johansen. Tracking visitors with adblockers. https://www.bjornjohansen.com/tracking-visitors-with-adblockers.

[169] Jason Bloomberg. Ad Blocking Battle Drives Disruptive Innovation. https://www.forbes.com/sites/jasonbloomberg/2017/02/18/ad-blocking-battle-drives-disruptive-innovation, 2017.

[170] Romain Cointepas. CNAME Cloaking, the dangerous disguise of third-party trackers. https://medium.com/nextdns/cname-cloaking-the-dangerous-disguise-of-third-party-trackers-195205dc522a, 2010.

[171] Daniel Plohmann, Khaled Yakdan, Michael Klatt, Johannes Bader, and Elmar Gerhards-Padilla. A Comprehensive Measurement Study of Domain Generating Malware. In *USENIX Security*, 2016.

[172] Lukasz Olejnik, Minh-Dung Tran, and Claude Castelluccia. Selling Off Privacy at Auction. In *NDSS*, 2014.

[173] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. In *CCS*, 2014.

[174] Daniel Hedin, Arnar Birgisson, Luciano Bello, and Andrei Sabelfeld. JSFlow: Tracking information flow in JavaScript and its APIs. In *SAC*, pages 1663–1671, 2014.

[175] Andrey Chudnov and David A Naumann. Inlined information flow monitoring for JavaScript. In *CCS*, 2015.

[176] Quan Chen and Alexandros Kapravelos. Mystique: Uncovering information leakage from browser extensions. In *CCS*, 2018.

[177] Steven Englehardt, Jeffrey Han, and Arvind Narayanan. I never signed up for this! Privacy implications of email tracking. *PETS*, 2018.

[178] treeinterpreter. https://pypi.org/project/treeinterpreter/.

[179] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *KDD*, 2018.

[180] Shifu Hou, Yujie Fan, Yiming Zhang, Yanfang Ye, Jingwei Lei, Wenqiang Wan, Jiabin Wang, Qi Xiong, and Fudong Shao. αCyber: Enhancing Robustness of Android Malware Detection System against Adversarial Attacks on Heterogeneous Graph based Model. In *IKM*, pages 609–618, 2019.

[181] Xiaoyun Wang, Joe Eaton, Cho-Jui Hsieh, and Felix Wu. Attack graph convolutional networks by adding fake nodes. *arXiv preprint arXiv:1810.10751*, 2018.

[182] Peter Snyder, Cynthia Taylor, and Chris Kanich. Most websites don't need to vibrate: A cost-benefit approach to improving browser security. In *CCS*, 2017.

[183] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. Fingerprinting the Fingerprinters:Learning to Detect Browser Fingerprinting Behaviors. In *S&P*, 2021.

[184] Anupam Das, Gunes Acar, Nikita Borisov, and Amogh Pradeep. The Web's Sixth Sense: A study of scripts accessing smartphone sensors. In *CCS*, 2018.

[185] Vasilios Mavroudis, Shuang Hao, Yanick Fratantonio, Federico Maggi, Christopher Kruegel, and Giovanni Vigna. On the privacy and security of the ultrasound ecosystem. *PETS*, 2017.

[186] Ben Stock, Sebastian Lekies, Tobias Mueller, Patrick Spiegel, and Martin Johns. Precise Client-side Protection against DOM-based Cross-Site Scripting. In *USENIX Security*, 2014.

[187] Sebastian Lekies, Ben Stock, and Martin Johns. 25 million flows later: Large-scale detection of DOM-based XSS. In *CCS*, 2013.

[188] M. Butkiewicz, H. V. Madhyastha, and V. Sekar. Understanding website complexity: measurements, metrics, and implications. In *IMC*, 2011.

[189] WebKit. Tracking Prevention in WebKit. https://webkit.org/tracking-prevention/, 2022.

[190] Enhanced Tracking Protection in Firefox for desktop. https://support.mozilla.org/en-US/kb/enhanced-tracking-protection-firefox-desktop, 2022.

[191] John Wilander. Bounce Tracking Protection. https://github.com/privacycg/proposals/issues/6, 2022.

[192] John Wilander. CNAME Cloaking and Bounce Tracking Defense. https://webkit.org/blog/11338/cname-cloaking-and-bounce-tracking-defense/, 2020.

[193] Brave Privacy Team. Fighting CNAME Trickery. https://brave.com/privacy-updates/6-cname-trickery/, 2020.

[194] Pouneh Nikkhah Bahrami, Umar Iqbal, and Zubair Shafiq. FP-Radar: Longitudinal Measurement and Early Detection of Browser Fingerprinting. In *Proceedings on Privacy Enhancing Technologies (PETS)*, 2022.

[195] Iskander Sanchez-Rola, Matteo Dell'Amico, , Davide Balzarotti, Pierre-Antoine Vervier, and Leyla Bilge. Journey to the center of the cookie ecosystem: Unraveling actors'; roles and relationships. In *S&P 2021, 42nd IEEE Symposium on Security & Privacy, 23-27 May 2021, San Francisco, CA, USA*, 2021.

[196] Brent Fulgham. Protecting Against HSTS Abuse. https://webkit.org/blog/8146/protecting-against-hsts-abuse/, 2018.

[197] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. Fp-stalker: Tracking browser fingerprint evolutions. In *IEEE Symposium on Security and Privacy (SP)*, 2018.

[198] Imane Fouad, Cristiana Santos, Arnaud Legout, and Nataliia Bielova. My Cookie is a phoenix: detection, measurement, and lawfulness of cookie respawning with browser fingerprinting. In *Privacy Enhancing Technologies Symposium (PETS)*, 2022.

[199] Quan Chen, Panagiotis Ilia, Michalis Polychronakis, and Alexandros Kapravelos. Cookie Swap Party: Abusing First-Party Cookies for Web Tracking. In *Proceedings of the Web Conference*, 2021.

[200] Sandra Siby, Umar Iqbal, Steven Englehardt, Zubair Shafiq, and Carmela Troncoso. WebGraph: Capturing Advertising and Tracking Information Flows for Robust Blocking. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, 2022.

[201] Xuehui Hu, Nishanth Sastry, and Mainack Mondal. CCCC: Corralling Cookies into Categories with CookieMonster. In *13th ACM Web Science Conference 2021*, page 234–242. Association for Computing Machinery, 2021.

[202] Dino Bollinger, Karel Kubicek, Carlos Cotrini, and David Basin. Automating Cookie Consent and GDPR Violation Detection. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, 2022.

[203] Lou Montulli. The Reasoning Behind Web Cookies. http://montulli.blogspot.com/2013/05/the-reasoning-behind-web-cookies.html, 2013.

[204] DoubleClick. https://web.archive.org/web/19970405225532/http://www.doubleclick.com/, 2022.

[205] L. Montulli D. Kristol. HTTP State Management Mechanism. https://datatracker.ietf.org/doc/html/rfc2109, 1997.

[206] This bug in your PC is a smart cookie. https://archive.org/details/FinancialTimes1996UKEnglish, 1996.

[207] Franziska Roesner, Tadayoshi Kohno, and David Wetherall. Detecting and Defending Against Third-Party Tracking on the Web. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 155–168, 2012.

[208] Aaron Cahn, Scott Alfeld, Paul Barford, and S. Muthukrishnan. An Empirical Study of Web Cookies. In *Proceedings of the 25th International Conference on World Wide Web*, page 891–901. International World Wide Web Conferences Steering Committee, 2016.

[209] Savino Dambra, Iskander Sanchez-Rola, Leyla Bilge, and Davide Balzarotti. When Sally Met Trackers: Web Tracking From the Users' Perspective. In *USENIX Security Symposium*, 2022.

[210] John Wilander. Intelligent Tracking Prevention. https://webkit.org/blog/7675/intelligent-tracking-prevention/, 2017.

[211] John Wilander. Intelligent Tracking Prevention 1.1. https://webkit.org/blog/8142/intelligent-tracking-prevention-1-1//, 2018.

[212] John Wilander. Intelligent Tracking Prevention 2.0. https://webkit.org/blog/8311/intelligent-tracking-prevention-2-0/, 2018.

[213] John Wilander. Intelligent Tracking Prevention 2.1. https://webkit.org/blog/8613/intelligent-tracking-prevention-2-1/, 2019.

[214] John Wilander. Intelligent Tracking Prevention 2.2. https://webkit.org/blog/8828/intelligent-tracking-prevention-2-2/, 2019.

[215] John Wilander. Intelligent Tracking Prevention 2.3. https://webkit.org/blog/9521/intelligent-tracking-prevention-2-3/, 2019.

[216] John Wilander. Full Third-Party Cookie Blocking and More. https://webkit.org/blog/10218/full-third-party-cookie-blocking-and-more/, 2020.

[217] Brendan Eich. C is for Cookie. https://brendaneich.com/2013/05/c-is-for-cookie/, 2013.

[218] Brendan Eich. The Cookie Clearinghouse. https://brendaneich.com/2013/06/the-cookie-clearinghouse/, 2013.

[219] Nick Nguyen. Latest Firefox Rolls Out Enhanced Tracking Protection. https://blog.mozilla.org/en/products/firefox/latest-firefox-rolls-out-enhanced-tracking-protection/, 2018.

[220] Disconnect tracking protection lists. https://disconnect.me/trackerprotection.

[221] Firefox rolls out total cookie protection by default to all users worldwide. https://blog.mozilla.org/en/products/firefox/firefox-rolls-out-total-cookie-protection-by-default-to-all-users-worldwide/.

[222] Internet Privacy with IE6 and P3P: A Summary of Findings. http://web.archive.org/web/20200731061208/http://www.spywarewarrior.com/uiuc/ie6-p3p.htm, 2001.

[223] Pedro Giovanni Leon, Lorrie Faith Cranor, Aleecia M McDonald, and Robert McGuire. Token attempt: the misrepresentation of website privacy policies through the misuse of p3p compact policy tokens. In *Proceedings of the 9th Annual ACM Workshop on Privacy in the Electronic Society*, 2010.

[224] Tracking Prevention in Microsoft Edge. https://docs.microsoft.com/en-us/microsoft-edge/web-platform/tracking-prevention, 2022.

[225] Justin Schuh. Building a more private web: A path towards making third party cookies obsolete. https://blog.chromium.org/2020/01/building-more-private-web-path-towards.html, 2020.

[226] Google. The Privacy Sandbox. https://developer.chrome.com/docs/privacy-sandbox/.

[227] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *2016 IEEE Symposium on Security and Privacy (SP)*, 2016.

[228] Sarah Sluis. Google Is Building Integrations For Publisher-Specific Identifiers. https://www.adexchanger.com/platforms/google-is-building-integrations-for-publisher-specific-identifiers/, 2021.

[229] About publisher provided identifiers. https://web.archive.org/web/20220614165742/https://support.google.com/admanager/answer/2880055?hl=en.

[230] Identity Guide. https://web.archive.org/web/20220115155115/https://yieldbird.com/identity-guide/.

[231] Ha Dao, Johan Mazel, and Kensuke Fukuda. CNAME Cloaking-Based Tracking on the Web: Characterization, Detection, and Protection. *IEEE Transactions on Network and Service Management*, 2021.

[232] Alessandra Van Veen and AP de Vries. Cookie Compliance of Dutch Hospital Websites. https://www.cs.ru.nl/bachelors-theses/2021/Alessandra_van_Veen___4683382___Cookie_Compliance_of_Dutch_Hospital_Websites.pdf, 2021.

[233] Waqar Aqeel, Balakrishnan Chandrasekaran, Anja Feldmann, and Bruce M Maggs. On landing and internal web pages: The strange case of jekyll and hyde in web performance measurement. In *Proceedings of the ACM Internet Measurement Conference*, 2020.

[234] Umar Iqbal, Charlie Wolfe, Charles Nguyen, Steven Englehardt, and Zubair Shafiq. Khaleesi: Breaker of Advertising and Tracking Request Chains. In *USENIX Security Symposium (USENIX)*, 2022.

[235] 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*, 2015.

[236] Audrey Randall, Peter Snyder, Alisha Ukani, Alex Snoeren, Geoff Voelker, Stefan Savage, and Aaron Schulman. Trackers bounce back: Measuring evasion of partitioned storage in the wild, 2022.

[237] Díaz-Morales and Roberto. Cross-Device Tracking: Matching Devices and Cookies. In *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, pages 1699–1704, 2015.

[238] Lotame lightning tag. https://web.archive.org/web/20220307010702/https://my.lotame.com/t/m1hxv7l/lotame-lightning-tag.

[239] Lotame Identity Resolution. https://web.archive.org/web/20220307153743/https://www.lotame.com/solutions/identity-resolution/. Accessed: 2022-06-18.

[240] Id5 identity cloud. https://web.archive.org/web/20220727094611/https://www.id5.io/identity-cloud/.

[241] ID5 - First Party IDs and Identity Resolution Methods Explained. https://web.archive.org/web/20220408035339/https://id5.io/news/index.php/2022/03/24/first-party-ids-and-identity-resolution-methods-explained/, 2022.

[242] Criteo Online Identification). https://web.archive.org/web/20220819071808/https://filecache.investorroom.com/mr5ir_criteo/977/download/Criteo_Online_Identification_May2020.pdf/.

[243] One Trust. Cookiepedia. https://cookiepedia.co.uk, 2022.

[244] Maximilian Hils, Daniel W Woods, and Rainer Böhme. Measuring the emergence of consent management on the web. In *Proceedings of the ACM Internet Measurement Conference*, 2020.

[245] Attentive cookie. https://docs.attentivemobile.com/pages/developer-guides/third-party-integrations/referral-marketing-platforms/talkable/, 2022.

[246] Omnisend cookie. https://support.omnisend.com/en/articles/1933402-explaining-and-managing-tracking-cookies, 2022.

[247] Hubspot cookie. https://knowledge.hubspot.com/reports/what-cookies-does-hubspot-set-in-a-visitor-s-browser, 2022.

[248] Google Analytics Cookie Usage on Websites). https://web.archive.org/web/20220812222800/https://developers.google.com/analytics/devguides/collection/gtagjs/cookie-usage.

[249] fbp and fbc Parameters. https://web.archive.org/web/20220722220344/https://developers.facebook.com/docs/marketing-api/conversions-api/parameters/fbp-and-fbc/.

[250] What facebook's first-party cookie means for adtech. https://web.archive.org/web/20220729210450/https://clearcode.cc/blog/facebook-first-party-cookie-adtech/.

[251] It's their word against their source code - tiktok report. https://internet2-0.com/whitepaper/its-their-word-against-their-source-code-tiktok-report/.

[252] Using cookies with tiktok pixel. https://web.archive.org/web/20220610074648/https://ads.tiktok.com/help/article?aid=10007540.

[253] Tiktok adds third-party cookies to its pixel – and tries to eat facebook's lunch. https://web.archive.org/web/20220623232016/https://www.adexchanger.com/online-advertising/tiktok-adds-third-party-cookies-to-its-pixel-and-tries-to-eat-facebooks-lunch/.

[254] Our new approach to address the rise of fingerprinting. https://blog.disconnect.me/our-new-approach-to-address-the-rise-of-fingerprinting/.

[255] Firefox's protection against fingerprinting. https://support.mozilla.org/en-US/kb/firefox-protection-against-fingerprinting.

[256] Cookieblock. https://github.com/dibollinger/CookieBlock.

[257] Dino Bollinger. Analyzing Cookies Compliance with the GDPR. https://www.research-collection.ethz.ch/handle/20.500.11850/477333. Thesis, ETH Zurich.

[258] uBlock Origin: Resources Library. https://github.com/gorhill/uBlock/wiki/Resources-Library#cookie-removerjs-.

[259] Cookies and the Experience Cloud Identity Service. https://experienceleague.adobe.com/docs/id-service/using/intro/cookies.html?lang=en, 2022.

[260] Understanding Calls to the Demdex Domain. https://experienceleague.adobe.com/docs/audience-manager/user-guide/reference/demdex-calls.html?lang=en, 2022.

[261] Code release for Encrypted DNS –> Privacy? A Traffic Analysis Perspective. https://github.com/spring-epfl/doh_traffic_analysis, 2020.

[262] Code release for WebGraph. https://github.com/spring-epfl/WebGraph, 2022.

[263] Report incorrectly removed content. https://forums.lanik.us/viewforum.php?f=64-report-incorrectly-removed-content, 2022.

[264] uBlockOrigin uAssets GitHub issues. https://github.com/uBlockOrigin/uAssets/issues?q=is%3Aissue+broken+, 2022.

[265] Michael Smith, Peter Snyder, Moritz Haller, Benjamin Livshits, Deian Stefan, and Hamed Haddadi. Blocked or broken? automatically detecting when privacy interventions break websites. *arXiv preprint arXiv:2203.03528*, 2022.

[266] Rahmadi Trimananda, Janus Varmarken, Athina Markopoulou, and Brian Demsky. Packet-level signatures for smart home devices. In *NDSS*, 2020.

[267] Hieu Le, Salma Elmalaki, Athina Markopoulou, and Zubair Shafiq. AutoFR: Automated Filter Rule Generation for Adblocking. In *NDSS*, 2022.

[268] Davis Yoshida and Jordan Boyd-Graber. Using Confusion Graphs to Understand Classifier Error. In *Proceedings of the Workshop on Human-Computer Question Answering*, pages 48–52, 2016.

[269] Lotame Detailed Reference Guide. https://web.archive.org/web/20210421084412/https://my.lotame.com/t/g9hxvnw/detailed-reference-guide. Accessed: 2022-06-18.

[270] Cartographer identity graph. https://web.archive.org/web/20220526085916/https://www.lotame.com/solutions/cartographer-identity-graph/.

[271] Panorama id. https://web.archive.org/web/20220327180718/https://www.lotame.com/panorama/id/.

SANDRA SIBY

+41 78 732 98 94 ⋄ sandra.siby@epfl.ch ⋄ https://sandrasiby.github.io/

EDUCATION

EPFL, Switzerland Sept 2017 – Sept 2022 (expected)
Doctoral Program in Computer Science (with EDIC Fellowship)
Security & Privacy Engineering Lab, advisor: Prof. Carmela Troncoso

ETH Zurich, Switzerland Sept 2012 – Sept 2014
Master of Science in Electrical Engineering & Information Technology
Specialization in Computers & Networks

National University of Singapore, Singapore Aug 2007 – July 2011
Bachelor of Engineering (Honors) in Electrical Engineering

York University, Canada Jan 2010 – April 2010
Student Exchange Programme

WORK EXPERIENCE

Mozilla Jun 2019 – Aug 2019
Privacy, Networking, and Security intern *USA*

· Research on developing a graph-based machine learning model for web tracking detection.
· Presented my work at the Mozilla Security Research Summit 2019.

Singapore University of Technology and Design Jun 2016 – Aug 2017
Research assistant, Research and Security Innovation Lab for IoT *Singapore*

· Developed a tool to scan and analyze communication among IoT devices.
· Studied the effectiveness of eavesdropping in newer WiFi standards.

Singapore University of Technology and Design Jan 2015 – April 2016
Research engineer, National Science Experiment project *Singapore*

· Helped develop a large scale wireless sensor network of about 50,000 nodes.
· Designed/implemented the server side architecture for data collection and analysis.
· Data visualization and web development.

J.P. Morgan July 2011 – Aug 2012
Technology Analyst (Corporate Development Program for fresh graduates) *Singapore*

· Provided Level 2 support, automated status report creation and conducted capacity analysis of all systems used by the commodities team.

Communications Lab, National University of Singapore June 2010 – Aug 2010
Research intern + continued as undergraduate thesis *Singapore*

· Developed a vehicular sensor system to gather environmental and location data.
· Analysed the possibility of lane sensing and inter vehicular communication to address traffic congestion.

Autodesk July 2009 – Dec 2009
Software development intern *Singapore*

· Developed features for AutoCAD 2010 and Project Cooper 2010 (C++).

· Participated in True Blue (an internal competition for defect fixing verification) and received the award for being the first person to verify 100 defects in Autodesk for AutoCAD 2010.

German University of Technology June 2008 – Aug 2008
IT assistant *Oman*

· Troubleshooting, documentation, network administration.

PUBLICATIONS/PATENTS

- **Conferences**

1. **WebGraph: Capturing Advertising and Tracking Information Flows for Robust Blocking**
   S. Siby, U. Iqbal, S. Englehardt, Z. Shafiq, C. Troncoso
   USENIX Security, 2022

2. **Encrypted DNS → Privacy? A Traffic Analysis Perspective**
   S. Siby, M. Juarez, C. Diaz, N. Vallina-Rodriguez, C. Troncoso
   Network and Distributed System Security Symposium (NDSS), 2020
   Also presented this work at the IETF 105 PEARG session (July 2019).

3. **Practical Evaluation of Passive COTS Eavesdropping in 802.11b/n/ac WLAN**
   D. Antonioli, S. Siby, N. O. Tippenhauer
   Cryptology And Network Security (CANS), 2017

4. **Link-Layer Device Type Classification on Encrypted Wireless Traffic with COTS Radios**
   R.R. Maiti, S. Siby, R. Sridharan, N. O. Tippenhauer
   European Symposium on Research in Computer Security (ESORICS), 2017

5. **METhoD: a Framework for the Emulation of a Delay Tolerant Network Scenario for Media-Content Distribution in Under-Served Regions**
   S. Siby, A. Galati, T. Bourchas, M. Olivares, S. Mangold, T.R. Gross
   International Conference on Computer Communications and Networks (ICCCN), 2015
   (Extended version in Advances in Computer Communications and Networks - from green, mobile, pervasive networking to big data computing (River Publisher), 2017)

6. **Mobile-Enabled Delay Tolerant Networking in Rural Developing Regions**
   A.Galati, T.Bourchas, S.Siby, S.Mangold, S.Frey, M.Olivares
   Global Humanitarian Technology Conference (GHTC), 2014

7. **ISSTA: An Integrated Sensing System for Transportation Applications**
   K.Moriuchi, S.Siby, Z.Hu, M.Motani
   Vehicular Technology Conference(VTC), 2011

- **Journals**

1. **Wearable Environmental Sensors and Infrastructure for Mobile Large-scale Urban Deployment**
   E. Wilhelm, S. Siby, Y. Zhou, X. J. S. Ashok, M. Jayasuriya, S. Foong, J. Kee, K. Wood, N. O. Tippenhauer
   IEEE Sensors, 2016

- **Workshops**

1. **DNS Privacy not so private: the traffic analysis perspective.**
   S. Siby, M. Juarez, N. Vallina-Rodriguez, C. Troncoso

Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETS 2018), co-located with PETS 2018

2. **IoTScanner: Detecting Privacy Threats in IoT Neighborhoods**
   S. Siby, R.R. Maiti, N. O. Tippenhauer
   International Workshop on IoT Privacy, Trust, and Security (IoTPTS 2017), co-located with AsiaCCS 2017

3. **System Architecture for Delay Tolerant Media Distribution for Rural South Africa**
   A.Galati, T.Bourchas, S.Siby, S.Mangold
   International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization (WiNTECH 2014)

- **Patents**

1. **Apparatus and method for monitoring a wireless network**
   N.O. Tippenhauer, R.R. Maiti, S. Siby, R. Sridharan

OTHER PROJECTS

Master thesis at Disney Research Zurich Feb 2014 - Sept 2014

· Developed and evaluated a delay tolerant media distribution system in rural South Africa, as part of MOSAIC2B (http://mobile-empowerment.org)

Paying for your Internet, one byte at a time Sept 2013 - Jan 2013

· Leveraged micropayment channels in the Bitcoin protocol for untrusted clients to pay an access point for Internet access.

TEACHING

Advanced Topics in Privacy Enhancing Technologies (CS-523)
Graduate course Spring 2021, Spring 2020
Computer Security (COM-301)
Undergraduate course Fall 2020, Fall 2019, Fall 2018
Information Security and Privacy (COM-402)
Graduate course Spring 2019, Spring 2018
Network Security (Lab teaching assistant at ETH Zurich)
Graduate course Fall 2013

TALKS

ZISC lunch seminar at ETH Zurich, 2022: WebGraph: Capturing Advertising and Tracking Information Flows for Robust Blocking
University of Lille Spirals seminar, 2022: WebGraph: Capturing Advertising and Tracking Information Flows for Robust Blocking
IETF 113 PEARG session, 2022: On the ineffectiveness of QUIC PADDING against website fingerprinting
UCL Infosec Seminar, 2022: WebGraph: Capturing Advertising and Tracking Information Flows for Robust Blocking
Microsoft Research confidential computing series, 2022: Towards developing effective defenses against website fingerprinting
Brave Research, 2021: WebGraph: Capturing Advertising and Tracking Information Flows for Robust Blocking

Ad Blocker Dev Summit, 2021: WebGraph: Capturing Advertising and Tracking Information Flows for Robust Blocking
Northeastern University, 2020: Encrypted DNS → Privacy? A Traffic Analysis Perspective
Mozilla Security Research Summit, 2019: A graph-based approach to tracker detection
IETF 105 PEARG session, 2019: Encrypted DNS → Privacy? A Traffic Analysis Perspective

SERVICE

Program Committee: PETS 2023, DNS Privacy Workshop (co-located with NDSS) 2021, IEEE ICBC 2019
External Reviewer: PETS, NDSS, USENIX Security

MENTORING

Project supervision: Supervised 14 student projects (3 master theses, 7 master semester projects, 4 bachelor semester projects)
Tech mentor: Blaze Accelerator program (https://www.epfl.ch/innovation/startup/blaze/)

TECHNICAL STRENGTHS

Computer Languages	Python*, Java†, C/C++†
Frameworks & Development Tools	Wireshark*, AWS*, Git*, Android SDK †
	(* – Currently using, † – Used in the past)

LANGUAGES

English	Fluent
Malayalam	Native
Hindi	Intermediate
French	Beginner (Level A2/B1)

ACTIVITIES

PhD Student representative for EPFL's Computer Science department 2018 – present

EPFL Computer Science PhD admissions committee 2021

Founder and member of *polygl0ts* – EPFL's CTF (Capture the Flag) team 2017 – 2020

Mentor for GirlsCoding (https://girlscoding.org/) 2018

Member of *kopipacket* – Singapore University of Technology and Design's CTF team 2016 – 2017

AWARDS

EPFL Computer Science department's Distinguished Service Awards 2020 and 2021 – for my contributions as the PhD student representative.

Singapore Mark Good Design Award 2016 – part of the team that was awarded for the SUTD NSE Virtual and Physical Interfaces Project.

Google Conference and Travel Grant 2014 – granted to women in computer science/engineering