KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Design and Evaluation of Website Fingerprinting Techniques

Marc Juárez Miró

Supervisor:
Prof. dr. Claudia Diaz

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

July 2019

Design and Evaluation of Website Fingerprinting Techniques

Marc JUÁREZ MIRÓ

Examination committee:
Prof. dr. Adhemar Bultheel, chair
Prof. dr. Claudia Diaz, supervisor
Prof. dr. ir. Bart Preneel
Prof. dr. ir. Frank Piessens
Prof. dr. Rachel Greenstadt
(NYU Tandon School of Engineering)
Prof. dr. Matthew Wright
(Rochester Insitute of Technology)

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

July 2019

Acknowledgements

The work included in this thesis would have not been possible without the support and assistance of many people. I want to express my gratitude to those who, in many different ways, have contributed to it.

First and foremost, I would like to express my deepest appreciation to my advisor, Prof. Claudia Diaz, for all the mentorship and support throughout my PhD. I will always be thankful for all the opportunities she gave me to develop as a researcher. She trusted me and steered me to work on the topic of this thesis, a topic that perfectly aligned with my research interests. Claudia shared with me her ideas, many of which are the basis for the work included in this thesis, and offered me to explore them with her. She was always kind and friendly; even during tough times, she was there to listen to me and guide me. Among all the advice she gave me, Claudia taught me about the importance of keeping the big picture in mind, a lesson that I treasure and try to practice as much as I can.

I would like to extend my sincere gratitude to Prof. Bart Preneel for his support and for providing such a vibrant research environment. He was always very encouraging and thoughtful in all his comments. I am deeply grateful to him, Prof. Frank Piessens, Prof. Matthew Wright, and Prof. Rachel Greenstadt, for their valuable feedback, which has substantially improved this thesis. I had the privilege to work with them for some of my research and, due to these collaborations, I had the opportunity to visit Rachel and Matt in Philadelphia and Rochester, respectively. I thank them for their warm hospitality and for taking such good care of me. Finally, I would like to thank Prof. Adhemar Bultheel for chairing this jury.

During these years I have shared my office with wonderful people. I keep good memories of the old ESAT office with Michael, Ero, Güneş, and Kai, during my Erasmus and the first years of PhD. In particular, I want to express my great appreciation to Güneş, who has not only been an amazing friend but also a

mentor to me; I admire his rigor and passion for research. In the new ESAT building, I also had the pleasure to share the office with Bekah and Chris. Bekah is one of the most easy-going people I have ever met. I especially appreciate that she introduced me to the world of beer-making and blue-grass music. Chris filled the room with complex concepts about philosophy, journalism, and Central Asia cultures, and came to the rescue by lending me the attire for my preliminary defense. Tariq deserves an honorary mention here since, although mine was not his official office, he was visiting it almost daily. I am indebted to him for all the advice and guidance he gave me. I am very lucky to have crossed paths with all these extraordinary people; each of them has had a great impact on me.

I am grateful to my colleagues in COSIC and, in particular, the Privacy Group for proof-reading my papers and listening to my presentations. I cannot thank them enough for all their feedback and support. Special thanks to (in no particular order): Iraklis, Rafa, Fatemeh, Tariq, Aysajan, Enrique, Elena, Josep, Sara, Mustafa, Abdel, Michael, Seda, Güneş, Bekah, Ilia, Ero, Edu, Ren, and Filipe. Especially helpful to me during thesis-writing time were Iraklis, Güneş, Fatemeh and Michael, who sent me their latex templates; and Sara, who was so kind to translate the abstract of this thesis to Dutch.

This journey would have not been as smooth as it was if it had not been for the excellent technical and administrative departments in ESAT. Special thanks to Péla, who was constantly anticipating anything I might need and whose cheerful emails made my days. Thanks also go to Elsy and Wim, who were extremely helpful with the financial administration, making me completely oblivious to it. I also want to acknowledge Dana's help with basically anything I asked her. Finally, the ESAT's IT group had a lot of patience with me. Special thanks to Rik for helping me maintain servers for our experiments.

I would also like to thank people in other institutions who guided me during these years. I am grateful to Dr. Sadia Afroz for giving me the opportunity to intern at ICSI, and Dr. Michael C. Tschantz, for his availability and his valuable feedback while I was there. I would like to thank my Google Summer of Code mentors, Mike Perry and Yawning Angel, for their knowledge and for introducing me to Tor's and Pluggable Transports' source code. I also would like to thank Roger Dingledine for being so welcoming and inviting me to the Tor-dev meetings. Finally, I want to thank Prof. Vicenç Torra for being so laid-back with me and for arranging my research stay in COSIC during my Erasmus. I am very grateful for all the opportunities he has given to me.

During my PhD I had the chance to work with brilliant researchers from all over the world. In fact, my research has been influenced by inspiring discussions I had with them. I would like to thank all of them and, in particular, I want to thank my co-authors (in no specific order): Jamie Hayes, Giovanni Cherubin,

Vera Rimmer, Tom Van Goethem, Mohsen Imani, Mike Perry, Sandra Siby, Carmela Troncoso, Narseo Vallina, Payap Sirinam, Rebekah Overdorf, Sadia Afroz, Steven Englehardt, Arvind Narayanan, Vicenç Torra, Rob Jansen, Rafael Gálvez, Tariq Elahi, Davy Preuveneers, Nick Nikiforakis, Seda Gürses, Güneş Acar, Frank Piessens, Rachel Greenstadt, Claudia Diaz, Bart Preneel, and Matthew Wright. It was a great pleasure to discuss and work with all of them.

All this research would have not been possible without the financial support of the FWO, who has generously funded four years of my studies and a research stay in the Rochester Institute of Technology.

I want to thank my friends at COSIC for all the fun activities and great times we have spent together: the COSIC weekends, the football matches, the conversations by the coffee machine, and the board games. Their friendliness made my daily-life in COSIC really enjoyable.

During my Erasmus, summer-schools, and internships, I made friends who have accompanied me during my PhD. I am particularly thankful to all the friends I had in Leuven: Iraklis, Dragoş, Güneş, Kai, Olesya, Marijke, Cristina, Abhay, Amogh, Sam, Ari, Imma, Tom, Chris, and Vera, among many others. Their warmth and care during these years made me feel at home.

I also wish to thank my friends in Ribes who, after all these years, still amuse me. During my time in Leuven they have let me know they are friends for life, despite the distance and the long time without seeing each other.

I would like to thank my parents and my sister for all their love and patience throughout these years, and for always being there for me. I extend my thanks to the rest of my family for their unconditional support and, especially, to my grandparents, for understanding why I have been so far from them all these years. Finally, I am lucky to have met Adriana and have had her with me for the last part of this journey. I want to thank her for all the patience and support she has devoted to me. Her love has made my life more meaningful.

Thank you all!

Marc Juárez Miró
Leuven, July 2019

Abstract

Website fingerprinting is a family of traffic analysis techniques that allows a local eavesdropper to infer which web pages a user visits over an encrypted channel. These techniques use machine learning classifiers to exploit features in the metadata of network traffic, such as the lengths and timing of network packets. Website fingerprinting has been shown to undermine the privacy provided by privacy enhancing technologies, such as HTTPS, encrypted proxies and VPNs, and even anonymous communications networks such as Tor.

This thesis compiles a series of studies on the methodology to evaluate the effectiveness of website fingerprinting techniques. The overarching contribution of these studies is to systematize the evaluation of website fingerprinting techniques, such that it provides an accurate assessment of the performance of website fingerprinting attacks and, consequently, informs the development of countermeasures to mitigate them.

Our first step was to review prior work and question assumptions made in the evaluation methodology of existing website fingerprinting attacks, arguing that such assumptions do not necessarily hold in practice: they are simplifying assumptions made in the laboratory but that might not hold in the real-world deployment of the attack. We assess the impact of the assumptions on attack effectiveness and show that the success of the attack substantially decreases when the assumptions do not hold.

We identify biases in the methods that prior work uses to evaluate the performance of the attack. First, we show that such evaluations have overlooked the effect of the websites' base rate on their measurements of prediction error. Second, prior work only measure the central tendency of classifier performance, albeit its high variance over different websites. We quantify the effect of such biases and conclude that they skew the view on the effectiveness of the attack, overestimating the adversary's success in practice.

On the same line of work, we have explored the threat model considered in previous studies, considering variants in order to uncover new threats that website fingerprinting techniques may enable. First, we show how website fingerprinting can be used from Tor middle nodes to find vulnerable targets and measure the popularity of Tor onion services. Second, we show that website fingerprinting is effective against DNS over HTTPS, a standardized protocol to encrypt DNS that is being widely adopted by providers such as Google and Cloudflare.

Turning to the development of defenses, we applied the results from the studies above to design a lightweight, network-level defense that eliminates latency overheads in the communication. We have specified the defense for Tor and evaluated it in realistic scenarios. Our evaluation shows that the defense is sufficiently effective in such scenarios, decreasing the attack's accuracy from 90% to 20%. This study contributed to the implementation of a generalization of this defense in Tor.

We design two novel defenses at the application layer: one as a browser add-on and the other running in the server-side. The latter is the first server-side countermeasure implemented against website fingerprinting. We show that these defenses are effective – the attack only attains 10% accuracy in the presence of the server-side defense. Moreover, they provide significant advantages with respect to network-level defenses: they are more efficient and easier to implement.

Finally, it was a central part of our work to study the traffic features exploited by the attack. In particular, we have mapped traffic features to high-level website characteristics that provide a better intuition of why the attacks are effective. We have elaborated our insights as guidelines that website maintainers can implement to protect their sites. Furthermore, we have developed a meta-learning classifier based on website characteristics that provides a user-friendly tool to estimate the exposure of sites to the attack. The Freedom of the Press has implemented our guidelines on SecureDrop, their whistle-blowing platform.

This thesis has contributed to define the threat that website fingerprinting poses to the privacy of web users by improving the accuracy of current evaluation methods, exploring the attack surface that website fingerprinting enables, and developing and testing countermeasures against it. As a result of this, our research has set in motion the implementation of defenses to protect Internet users against website fingerprinting in real-world systems such as Tor, SecureDrop, Google, Cloudflare, and Mozilla.

Beknopte samenvatting

Website fingerprinting is een verzameling van netwerkanalysetechnieken waarmee een lokale afluisteraar kan afleiden welke webpagina's een gebruiker bezoekt via een versleuteld kanaal. Deze technieken gebruiken classificatie op basis van machinaal leren om te profiteren van eigenschappen van metagegevens van het netwerkverkeer, zoals de lengte en timing van netwerkpakketten. Van website fingerprinting is aangetoond dat het de privacy ondermijnt die geboden wordt door privacybevorderende technologieën, zoals HTTPS, versleutelde proxy's en VPN's, en zelfs anonieme communicatienetwerken zoals Tor.

Dit proefschrift bundelt een reeks studies over de methodologie die gebruikt wordt om de effectiviteit van website fingerprinting technieken te evalueren. De overkoepelende bijdrage van deze studies is dat ze de evaluatie van website fingerprinting technieken systematiseren, wat een accurate beoordeling van de performantie van website fingerprinting aanvallen toelaat, en bijgevolg de ontwikkeling van tegenmaatregelen om ze te verhinderen informeert.

Onze eerste stap was het beoordelen van bestaand werk en het in vraag stellen van aannames in de evaluatiemethodes van bestaande website fingerprinting aanvallen. We argumenteren dat zulke aannames niet noodzakelijk gelden in realiteit: het zijn vereenvoudigingen, gemaakt in een proefomgeving, die niet noodzakelijk van toepassing zijn bij een uitrol in de echte wereld. We beoordelen de impact van deze aannames op de effectiviteit van de aanvallen en tonen aan dat het succes van de aanval aanzienlijk afneemt als de aannames niet gelden.

We identificeren vooroordelen in de methoden die eerder werk gebruikt om de prestatie van een aanval te evalueren. Ten eerste tonen we aan dat zulke evaluaties het effect van de base rate van de website op hun meting van de voorspellingsfout over het hoofd zien. Ten tweede meet eerder werk enkel de centrale tendens van classificatieperformantie, ondanks zijn hoge variantie over verschillende websites. We kwantificeren het effect van zulke vooroordelen en

concluderen dat ze de visie op de effectiviteit van de aanval scheeftrekken, waardoor het succes van de tegenstander in de praktijk wordt overschat.

Daarenboven hebben we het bedreigingsmodel onderzocht dat gebruikt wordt in eerdere studies. Hierbij hebben we verschillende varianten beschouwd om nieuwe bedreigingen te ontdekken die mogelijk gemaakt worden door website fingerprinting technieken. Ten eerste laten we zien hoe website fingerprinting gebruikt kan worden vanuit Tor middenknopen om kwetsbare doelwitten te vinden en de populariteit van Tor onion-diensten te meten. Ten tweede tonen we aan dat website fingerprinting doeltreffend is tegen DNS over HTTPS, een gestandaardiseerd protocol om DNS te versleutelen dat momenteel op grote schaal geadopteerd wordt door providers zoals Google en Cloudflare.

Wat het ontwikkelen van verdedigingsmechanismen betreft, hebben we de resultaten uit bovenstaande onderzoeken toegepast om een lichtgewicht verdedigingsmechanisme op netwerkniveau te ontwikkelen dat latency overhead in de communicatie elimineert. We hebben dit verdedigingsmechanisme gespecifieerd voor Tor en geëvalueerd in realistische scenario's. Uit onze evaluatie blijkt dat het verdedigingsmechanisme voldoende effectief is in zulke scenario's, de nauwkeurigheid van de aanval neemt af van 90% tot 20%. Deze studie heeft bijgedragen aan de implementatie van een veralgemening van dit verdedigingsmechanisme in Tor.

We ontwerpen twee nieuwe verdedigingsmechanismen op de applicatielaag: één als browser add-on, en een andere op de server. Deze laatste is de eerste tegenmaatregel tegen website fingerprinting die geïmplementeerd is op de server. We tonen aan dat deze verdedigingsmechanismen doeltreffend zijn – de aanval bereikt slechts 10% nauwkeurigheid in de aanwezigheid van ons server-side verdedigingsmechanisme. Bovendien bieden ze aanzienlijke voordelen tegenover netwerk-niveau verdedigingsmechanismen: ze zijn efficiënter en eenvoudiger te implementeren.

Ten slotte was een centraal onderdeel van ons werk het bestuderen van eigenschappen van netwerkverkeer die benut worden door de aanval. Meer specifiek hebben we deze netwerkverkeerseigenschappen gemapt op high-level website-eigenschappen die ons een betere intuïtie verschaffen over waarom de aanvallen effectief zijn. We hebben onze inzichten uitgewerkt als richtlijnen die websitebeheerders kunnen implementeren om hun sites te beschermen. Bovendien hebben we een meta-lerend classificatie-algoritme ontwikkeld, gebaseerd op website-eigenschappen, dat ons een gebruiksvriendelijke tool verschaft om de blootstelling van sites aan de aanval in te schatten. Freedom of the Press heeft onze richtlijnen geïmplementeerd op SecureDrop, hun klokkenluidersplatform.

Dit proefschrift heeft bijgedragen aan het definiëren van de bedreiging die website fingerprinting vormt voor de privacy van internetgebruikers. Ten eerste door de nauwkeurigheid van de huidige evaluatiemethoden te verbeteren, ten tweede door het aanvalsoppervlak te onderzoeken dat mogelijk gemaakt wordt door website fingerprinting en ten slotte door het ontwerpen en testen van tegenmaatregelen. Daardoor heeft ons onderzoek de implementatie gestart van verdedigingsmechanismen om internetgebruikers te beschermen tegen website fingerprinting in reële systemen zoals Tor, SecureDrop, Google, Cloudflare en Mozilla.

Contents

List of Figures

List of Tables

List of Abbreviations

ALPaCA Application Layer Padding Concerns Adversaries.

API Application Programming Interface.

AS Autonomous System.

CDN Content Delivery Network.

CMS Content Management System.

CSS Cascaded Style Sheets.

DNS Domain Name System.

FPR False Positive Rate.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IP Internet Protocol.

ISP Internet Service Provider.

LLaMA Lightweight application-Layer Masquerading Add-on.

SVM Support Vector Machine.

TCP Transmission Control Protocol.

TPR True Positive Rate.

VM Virtual Machine.

VPN Virtual Private Network.

WF Website Fingerprinting.

WTF-PAD Website Traffic Fingerprinting Protection with Adaptive Padding Defense.

Part I

Design and Evaluation of Website Fingerprinting Techniques

Chapter 1

Introduction

The widespread adoption of the Internet and the applications that it supports has raised unprecedented privacy concerns. With more than 50% of the world population connected to it [75], the Internet is not only the preferred medium for most electronic communication, but it has also enabled new forms of interaction, fundamentally changing our society and the way we communicate. Yet, despite our society's dependency on the Internet, the technologies that compose it are still vulnerable to a wide range of privacy threats.

One such threat stems from the analysis of communications *metadata*, also known as *traffic analysis*. Metadata, such as timing and volume of messages, can compromise private information of the communication, even if the messages have been encrypted. Studies have shown that metadata leak information about: VoIP conversations [81], the passwords of SSH sessions [70], or the identity of devices across different networks [12]. Therefore, traffic analysis undermines the confidentiality properties provided by the use of encryption.

As data packets travel through the Internet, their metadata are susceptible to inspection by a wide range of entities, from local network eavesdroppers to infrastructure providers and governments. The "Snowden revelations" uncovered governmental intelligence-gathering programs that indiscriminately extract and process Internet metadata [3], and go as far as to tap the submarine cables that funnel most of the world's Internet communications [2]. The potential application of these programs for surveillance raises serious privacy concerns.

Anonymous communications systems strive to protect against traffic analysis from powerful adversaries such as the ones above. For instance, *mix* [16] and *onion routing* [72] networks hide some of the metadata from network

eavesdroppers to provide anonymity at the Internet layer. Tor is the most widely used onion routing network as, unlike mixes, it does not add delay to communication, allowing for interactive applications such as web browsing [21]. However, not mixing the traffic comes at a cost in privacy; due to its low-latency constraints, Tor is more vulnerable to traffic analysis than mix networks [52].

In particular, several studies have shown that Tor's anonymity properties are compromised by a traffic analysis technique known as *website fingerprinting* (WF). WF allows a passive and local adversary to identify the web resources being accessed by Web users. This technique exploits patterns that are observable in the metadata of encrypted traffic and are unique to each specific web resource. Prior work has shown that WF is effective in identifying web resources being accessed over HTTPS [17, 49], SSH [43], VPNs [29], web proxies [30, 71], and even the Tor network [15, 57].

The goal of this thesis is twofold: first, to precisely assess the threat that WF techniques pose for Web users; and, second, to develop countermeasures to protect web users against WF attacks. We have focused on WF in the context of Tor, hence in most of our work we have studied WF attacks applied to Tor and developed countermeasures specifically designed for it. However, our analysis is not restricted to Tor and also includes other privacy enhancing technologies such as HTTPS and encrypted DNS.

In order to achieve these goals, we explore the WF threat model, considering variations of the threat model found in the literature. This allows us to uncover new threats and thus understand in depth the impact of this attack. In addition, we critically examine evaluations of WF techniques performed by prior work, exposing biases in their methodology and suggesting methods to mitigate such biases, in order to more accurately determine the effectiveness of website fingerprinting techniques in practice.

1.1 Summary of contributions

The overarching contribution of this work is to provide a more rigorous modeling and evaluation of WF techniques, which is crucial to determine the practical effectiveness of the attacks and, consequently, inform the design of defenses against them. The outcome of this work has had impact on real-world privacy-enhancing technologies such as Tor. Our work has raised awareness of the WF problem in the academic community and has spurred research in the field, motivating several follow-up studies. Moreover, some of the contributions of our work apply to other fields where traffic analysis is used, such as intrusion detection systems and censorship, which benefit from some of these contributions.

The following is a summary of the contributions of this thesis:

Realistic evaluations of WF techniques. We critically review evaluations of WF attacks in prior work and identify unrealistic assumptions in their methodology [35]. We experimentally assess the impact of such assumptions on the effectiveness of the attack. Similarly, we identify flaws in the evaluation methodology that introduce a bias in the estimations of the attack's success rate [35, 55]. We propose new evaluation methods that allow to detect and assess such biases and incorporate them in the evaluation methodology [36, 68].

Our evaluations provide a more realistic view of the attacks which is paramount for the development of future countermeasures: having a well-defined bound of the success rate of the attack allows defenses to optimize resources accordingly. In fact, our results show that previous evaluations had overestimated the attack's average success rate in practice, implying that, in general, adequate defenses could be achieved at a lower cost than previously thought.

I am the main contributor of the work on the realistic evaluation of WF techniques. This work has been published in CCS'14 [35], CCS'17 [55] and ESORICS'16 [36], and submitted to USENIX'20 [68].

Identification and assessment of new threats. We explore the WF threat model by discovering attacks enabled by WF that had not yet been considered in the literature [32, 68]. In particular, we study the use of WF to identify Tor Onion Services [19, 32, 55], to select potential targets from strategic vantage points [32], to measure the popularity of onion services [32], and to deploy real-time web censorship in encrypted DNS and HTTPS connections [68]. We developed the attacks and the methodology to study the effectiveness of WF techniques in such scenarios.

Our work unveils attack vectors enabled by WF techniques and determines the severity of such attacks. Part of the attack surface can be tackled by defenses that protect against WF in the standard threat model, but some of the attacks require standalone countermeasures. We have responsibly disclosed our findings to service providers that are vulnerable to such attacks.

The threats that we uncovered for onion services were studied in our work presented in NDSS'18 [32]. The work on applying fingerprinting techniques to encrypted DNS traffic is under submission to USENIX'20. I am a principal author and made substantial contribution to both publications.

Engineering and analysis of traffic features. The traffic features that WF exploits are a key component of WF techniques. A transversal contribution of our work has been to advance the methodology to select such features. One

of our first contributions on this line of work is a classifier-independent feature analysis of WF techniques that allows to map high-level website characteristics, such as the presence of advertisements or tracking scripts, with low-level traffic features, such as the timing and size of network packets [55]. Based on the results of such analysis we derive design guidelines that can help website maintainers protect their users against WF [55].

Freedom of the Press Foundation engineers modified the SecureDrop webpage template to implement some of our recommendations.

In addition, we designed a meta-classifier that can predict the vulnerability of a website to WF only based on the aforementioned high-level characteristics. Such method could be used by website developers to estimate the exposure of their websites to the attack without having to collect network traffic and train a classifier.

Finally, we propose an information-theoretic framework to evaluate the feasibility of WF for the purpose of censorship on encrypted DNS traffic. This framework provides an upper bound for the performance of any WF attack that is independent of the specific classifier or features that the attacker uses [68].

The work that has contributed to the engineering and analysis of traffic features has been published in CCS'17 [55], NDSS'18 [32], and submitted to USENIX'20 [68]. I am the main contributor of the analysis of variance in classification error presented in [55] and the entropy analysis in [68]. Moreover, I have developed the feature sets presented in [32, 68].

Novel and practical defenses against WF attacks. First, we have proposed WTF-PAD [36], a countermeasure that has been specifically designed to be implemented in Tor. WTF-PAD is based on Adaptive Padding [67], a network-level traffic analysis countermeasure that minimizes latency added by cover traffic. Such low latency performance is specially suited for low-latency anonymous communications systems such as Tor. We have adapted it to protect Tor against WF and evaluated its effectiveness in realistic scenarios.

Second, we have proposed two application-level defenses [19]. We argue that countermeasures that operate at the application layer have the advantage of having access to the latent features that WF exploits, namely, the sizes of web resources that are observed indirectly from the size of network packets. This approach offers defense strategies that are not possible at the network layer. Within this work, we proposed a lightweight client-side countermeasure and implemented, to the best of our knowledge, the first server-side defense against WF, having Tor onion services in mind as the main use-case.

Our work has contributed to the implementation of a general framework for the development of traffic analysis defenses in Tor [47]. Furthermore, we have collaborated with the Freedom of the Press Foundation to implement a prototype of our server-side defense to be used in their SecureDrop platform [1, 6].

The work on WTF-PAD is published in ESORICS'16 [36] and the application-level defenses are published in PETS'17 [19]. I am the main contributor of the work on developing and evaluating WTF-PAD. The work on application-level defenses is shared among all co-authors. In particular, I had an important role in the development and evaluation of the client-side defense.

Large-scale automated collection of traffic datasets.

We have collected the largest publicly available datasets of traffic data for Tor Onion services [19, 55] and the largest publicly available dataset for regular web traffic data [64]. We have also collected the first dataset of encrypted DNS traffic data for WF [68].

The software to automate data collection for traffic analysis is also part of our contributions to help systematize large scale WF studies. Our crawlers drive the Tor Browser in order to simulate the browsing behavior of a Tor user. Moreover, one of our crawlers collects data from other positions in the circuit while being compliant with the Tor ethical guidelines [32].

We have engaged in the practice of reproducible research by making all software and datasets produced during our studies available to other researchers. Other research groups have used our tools to collect their own datasets [42, 54, 63]. We have provided assistance to other researchers in using our software and data, and we are willing to continue to do so.

I contributed substantially to the implementation of the crawlers and the collection of datasets in all the studies included in this thesis.

1.2 Other contributions

The work included in this thesis is a selection of the contributions that we have made to the field of computer security and privacy. We have selected these articles because they are relevant to the topic of this thesis. However, the research we have conducted in other areas of the field has influenced the work presented in this thesis in one way or another. The following is a summary of these other contributions in reverse chronological order.

Website fingerprinting techniques based on deep learning We have applied deep learning techniques to develop WF attacks [64, 69]. Deep learning is a family of machine learning classifiers based on multi-layer artificial neural networks. We evaluated the suitability of different deep learning algorithms for WF and were able to obtain accuracies of up to 98% under the same conditions as prior work [69]. We also evaluate an open world of pages and measure the data distribution shift over time [64]. In addition, we evaluate network-level WF defenses to the attacks and show that the attacks defeat previous defenses in the lab, including one of the defenses we propose in the work compiled in this thesis.

This work was published in NDSS'18 [64] and CCS'18 [69]. I have provided key ideas in both works and contributed to the writing of the text.

Evaluation of web censorship measurement studies Similarly to how we challenge assumptions in the methodology to evaluate WF attacks, we have reviewed the methodology of web censorship measurement studies. We looked for assumptions these studies made that might have introduced biases in the results of their measurements. For instance, many of these studies deploy measurement probes exclusively in university networks such as PlanetLab. Since university networks are often more privileged than regular home networks, placing probes only on university networks might have introduced a selection bias, underestimating the actual scope of censorship.

This work was published in OPERANDI, a workshop co-located with PETS'18 [10]. I contributed to the development and collection of the datasets, and the analysis of the data.

Profiling of Tor users We have explored profiling techniques enabled by WF techniques. In this work we considered an adversary who only uses unlabeled observations. Unlike WF, such data cannot be used to identify the pages that a user is visiting. We show that these data can be useful to profile users by clustering them into groups of interest. The adversary can use these groups to select potential targets, e.g., users that visit uncommon sites and whose profiles stand out from the crowd. The adversary can then allocate resources to mount more sophisticated attacks against them. We showed in the paper that, for a world of 100 sites, an adversary is able to group visits of different users to the same site with more than 50% success rate without any training data.

This work was published at INFER, a workshop co-located with PETS'16 [24]. I contributed to the evaluation of the classifier and the writing of the text.

Web tracking measurements. We have performed measurements of the prevalence of various tracking technologies on the Web [8,9]. First, we developed

a tool, that we call "FPDetective" [9], to conduct a large-scale, automated measurement of the prevalence of browser fingerprinting. We found that more than 45 of the top 10K most popular sites and 404 out of the top 1M use Flash-based and JavaScript-based fingerprinting, respectively. We evaluated existing countermeasures against fingerprinting and uncovered several vulnerabilities in the Tor Browser.

As a follow-up study we measured the prevalence of advanced tracking mechanisms such as canvas fingerprinting, cookie syncing and evercookies [8]. We contributed to the development of OpenWPM [4], a measurement toolkit that we used to automate the visits to the top most popular 100,000 websites and detect tracking scripts. We found that 5% of the sites were using canvas fingerprinting, including a popular plugin that was embedded in the official website of the White House. We also identified the ten top parties involved in cookie syncing and showed that they can link 40% of the browsing history of a user. Regarding evercookies, we found that 107 out of the 10,000 most popular sites use evercookies to respawn HTTP cookies. These studies had considerable media coverage in several international online newspapers.

The results from the former study were published in CCS'13 [9] and the results for the latter were published in CCS'14 [8]. For the former study, I contributed significantly in the development of "FPDetective" and the collection and analysis of the data. Regarding the study on advanced fingerprinting techniques, my main contribution was in testing the implementation of the canvas fingerprinting detection mechanisms.

Private web search We designed and evaluated an intelligent agent, implemented as a browser add-on, that proxies queries to a search engine [37–39]. The agent creates different profiles for the same user and manages the user's tracking identifiers, such that queries that are related semantically (e.g., they are part of the same topic) are logged in the same profile. This strategy decreases the risk of de-anonymization while preserving a certain degree of search personalization.

This work led to an article in the International Journal of Intelligent Systems [38], a follow up study in the the International conference on Privacy, Security and Trust [37], and a publication in a book [39]. I am the main author of this work.

1.3 Structure

This thesis is divided into two parts. Part I begins with an introduction to the topic and the contributions of the thesis. This introduction is followed

by Chapter 2, that provides the necessary background for the reader to understand the contributions of the thesis, and Chapter 3, that contextualizes the contributions of the thesis. Finally, Chapter 4 concludes with a synthesis of our work and avenues for future research.

Part II comprises the six publications compiled in this thesis:

1. A study that reviews and evaluates assumptions made on previous evaluations of WF attacks [35].
2. The design and evaluation of a WF defense specifically designed for Tor [36].
3. A study of application-level defenses for Tor Onion services [19].
4. A per-website feature analysis study of WF attacks on Tor onion services [55].
5. Study of WF threat models from a middle node [32].
6. A new WF technique that exclusively uses encrypted DNS traffic [68].

Chapter 2

Preliminaries

Website Fingerprinting (WF) is a traffic analysis attack that allows a network-level adversary to infer the browsing history of web users by just analysing the metadata of their encrypted network traffic. The intuition behind the attack is that differences in embedded content (e.g., distinct images, scripts, styles) can be measured in the timing and sizes of network packets, even if the packets have been encrypted or anonymized.

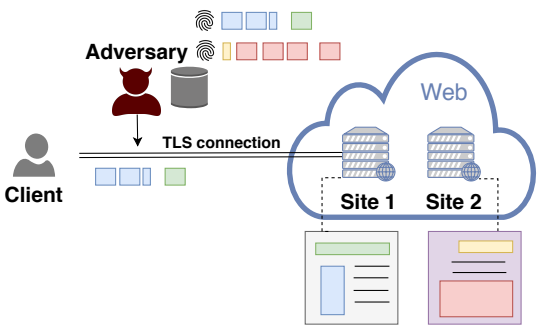


Figure 2.1: Two web pages with resources of different sizes. The sizes of such resources can be observed in the traffic by analyzing the sizes of network packets. Such information allows an adversary to distinguish between the pages.

Such differences are unique to each web page and can be distinguished in the traffic, allowing a network eavesdropper to identify remote content being accessed by a user.

For example, suppose that a page has an image with an uncommon size; when we download the image, it will result in an uncommon sequence of packet sizes. Hence, the observation of packet size sequences will leak information about the web page. This is illustrated for two different pages in Figure 2.1

The first studies to use packet sizes to identify web pages were published in the late nineties, showing that although SSL provided confidentiality, it did not hide the size of page resources [18, 50]. Even though these first studies used rudimentary techniques and were limited to a dozen web pages, they spurred research on WF. The studies that followed adopted statistical classifiers to re-identify page visits under different privacy-enhancing technologies such as: stripping, encrypted proxies [30, 71], SSH tunnels [43], encrypted VPNs and anonymous communication systems such as JAP and Tor [29].

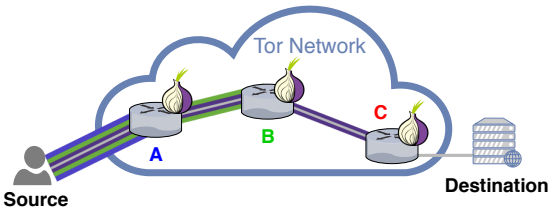
In this thesis we have focused on Tor because it is, from the privacy enhancing technologies listed above, the one that provides stronger privacy guarantees. In the following section we in introduce Tor and its threat model.

2.1 The Onion Router (Tor)

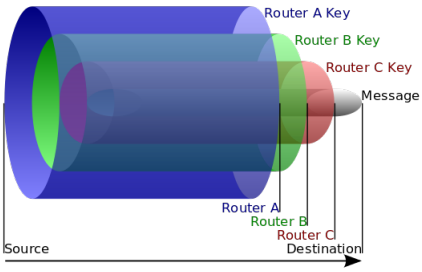
With more than two million daily users [74], Tor has emerged as the most widely used anonymous communication system to browse the Web. Tor is an overlay network, i.e., a network whose nodes are connected over TCP/IP. Tor nodes are spread all over the world, and are run and maintained by volunteers. Such volunteer-based model fosters diversity in the network and distributes trust, as opposed to having a few entities maintaining and controlling the nodes.

Tor is a source-routed anonymous communications system, meaning that clients choose the path of nodes, or *circuit*, over which their messages will be routed. The clients encrypt messages in as many layers as there are nodes in the circuit. Each layer is encrypted using a key for a different node following the principles of onion routing [72] (see Figure 2.2), such that, at each hop, only one node can decrypt and forward the message. This way, each of the nodes learns the addresses of the adjacent nodes in the circuit, but none of the nodes can learn both the origin and the destination of the communication – if the circuit has more than one node.

Threat model
Tor's original threat model considers local adversaries and explicitly acknowledges that it excludes adversaries who can observe both ends of the communication [21]. Indeed, if the adversary owns the first and last node in a circuit, since Tor does not hide timing metadata, the adversary can correlate



(a) A Tor circuit from source to destination.



(b) Each of the keys used to encrypt the layers of a Tor circuit. Credit: Harrison Neal, distributed under CC-BY-SA-3.0 (unmodified)

Figure 2.2: The onion routing protocol. The figure above shows the layers of a circuit from the client to the web server. The figure below provides more detail about the layers, indicating the key used to encrypt each layer

connections entering the network with connections that leave it by matching their volume and timing [34]. This is known as *end-to-end correlation* and it requires a more powerful adversary than WF. In order to limit end-to-end correlation, circuits in Tor are by default three-node long. This way the nodes in the edges of the path cannot directly observe each other's IP addresses – and thus trivially locate each other.

Guard nodes

The first node of a circuit is called the *guard* node. Since guards have direct communication with the client, Tor puts more trust in them than in the rest of the circuit. In particular, clients will select a node to be the first node of all the circuits they build for a period of four months.

The rationale for pinning the guard is to reduce the probability of selecting a compromised node over time. Tor is constantly building circuits in the background and uses a different circuit for every domain name that it connects

to. If a client selected a different guard for every circuit, it would rapidly hit a compromised node [23]. Pinning the guard slows down an adversary that aims at attacking a specific user. This design decision is based on the assumption that an adversary whose nodes are always being selected by the same set of users has diminishing returns. Without that assumption, the adversary would repeatedly attack a few users, which is not necessarily fair.

Since a low-bandwidth guard negatively affects the performance of a client for a long time, nodes that are selected as guards must meet a number of requirements in bandwidth and up time. Once a node satisfies those requirements, it earns the *guard* flag, indicating that it can be selected as such.

Tor's implementation is based on a SOCKS proxy, allowing to route TCP-based protocols other than HTTP(S). The Tor software is divided into two main components: the Tor Browser, a "hardened" fork of Firefox; and the *onion router*, which routes Tor messages, also known as *cells*, through the Tor network. There is an onion router running at the client and each of the relays that compose the Tor network. The Tor Browser is especially relevant for WF, as it is the recommended way of accessing websites over Tor. We explain it in more detail in the next section.

WF in Tor

As described above, Tor is designed to protect its users against adversaries who own a fraction of nodes in the network and. In particular, it should protect against passive network eavesdroppers who observe the traffic between the client and the entry, hence it should protect against WF. However, Tor's threat model did not take into account attacks based on machine learning classifiers such as WF. Research on WF has provided evidence that Tor is, to some degree, vulnerable to WF attacks.

In Figure 2.3, we depict the progression of WF classifier accuracy over the last decade. As we see in the figure, even though the first WF attack that was applied against Tor achieved less than 3% success rate [29], three years later, Panchenko et al. presented a refined version of the attack with an average 55% success rate [57]. Panchenko et al.'s work was succeeded by a series of studies that, with features specifically designed for Tor traffic and more advanced machine learning techniques, claimed success rates of over 90% [15, 28, 56, 77, 78]. Many of such studies portray WF as a devastating attack against Tor, completely defeating Tor's purpose.

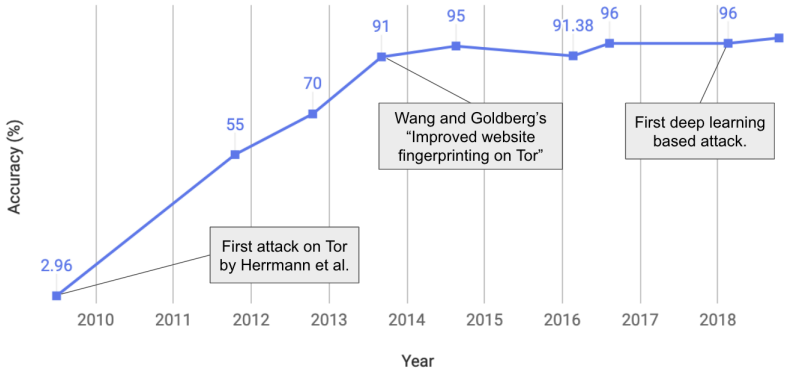


Figure 2.3: Timeline of WF attacks in the last decade and their reported accuracies. In 2009 Herrmann et al. propose the first attack against Tor [29]. In 2013, Wang and Goldberg present, in "Improved website fingerprinting on Tor", an attack that achieved more than 90% accuracy [78]. In 2018, Rimmer et al. present the first deep-learning based attack against Tor [64].

2.1.1 Tor Browser

Tor is shipped in a bundle that includes its own browser, a modified version of Firefox that aims to protect Tor users from a number of different adversaries [61]. Similarly to Firefox's incognito mode, the Tor Browser does not keep state across different browser sessions: it wipes out cookies, browsing history, and web storage on every new session. The Tor browser goes beyond incognito mode in protecting the privacy of Tor users in many aspects. First, it disables third-party cookies by default to shrink the tracking surface; second, it comes with a number of extensions that protect the user from web security vulnerabilities: HTTPS everywhere preemptively fetches the HTTPS version of visited pages, the Tor button ensures that traffic is effectively routed through the Tor network, and NoScript blocks malicious Flash and JavaScript scripts.

Moreover, the Tor Browser implements countermeasures against *browser* fingerprinting, which should not be confused with WF. In browser (or device) fingerprinting a malicious website identifies and tracks users through unique identifiers that are extracted from the software, hardware and configuration of the user's device [7, 9]. The Tor Browser aims at having a single, stable fingerprint for all Tor users. In particular, it warns against maximizing the browser window and bundles a set of fonts with the browser that are used in place of the fonts installed in the operating system.

Randomized Pipelining.
The Tor Browser also features a WF countermeasure called *Randomized Pipelining* (RP) [58]. RP was envisioned as a lightweight defense against WF [61]. It consists in queuing and randomizing the requests in the HTTP pipeline. However, RP has been proven to be ineffective against WF attacks in several evaluations [15, 35, 78].

2.1.2 Tor Onion services

Tor Onion services, formerly known as hidden services, are special web servers that can only be accessed over the onion service protocol without having to reveal their IP address to the users. It thus provides location privacy and network-level anonymity to the web server towards the users that visit it. Since onion services are not indexed by regular search engines, they form part of what is popularly known as the *deep* web.

A use case of Onion Services is whistle-blowing: since the server is more difficult to locate geographically, onion services are, to a certain extent, resistant to compulsion attacks, and thus protect content publishers from retaliation. One example of this is the Freedom of the Press Foundation's SecureDrop [5], a web platform that runs as an onion service, used by popular media outlets such as The New York Times to protect journalists and their sources. Other popular examples are WikiLeaks and GlobalLeaks whose official websites are hosted as onion services.

The Onion service protocol.
The protocol used to contact onion services is different from the one used to access regular websites. The onion service protocol allows a client and a server to agree on a Tor node, called *Rendezvous Point*, without having to rely on IP addresses. The Rendezvous Point, as its name indicates, acts as a "meeting point" to which both client and server build a Tor circuit.

First, as illustrated in Figure 2.4, in order to make the server reachable as an onion service, it builds a number of Tor circuits to be kept reachable as long as the service is available (1) – although in practice, these circuits rotate periodically. The exit nodes of these circuits are called *Introduction Points*, and their addresses, along with the onion service public key, form the onion service *descriptor*. Then, the onion service announces itself to the Tor network by publishing the descriptor in a database called *HS Directory* (2), implemented as a distributed table over several nodes of the network.

Clients identify onion services by their onion addresses. The onion address is derived from the onion service's public key and communicated to the user

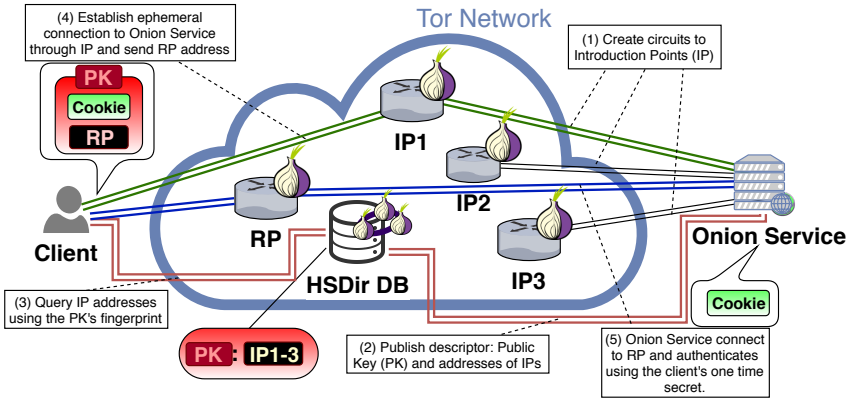


Figure 2.4: The Tor Onion Service protocol. Tor circuits are represented by double lines.

out-of-band (e.g., published in a public index). When a client connects to the onion service, she first builds a Tor circuit. The exit node of that circuit is the Rendezvous Point. Then, she uses the onion address to look up the onion service's Introduction Points in the HS Directory (3). Note that connections to the HS Directory are also done over Tor. Finally, she connects to an Introduction Point and sends the address of the Rendezvous Point to the onion service (4), who will build a circuit to it in order to "meet" with the client (5). At this point, there is a six-hop circuit between the client and the onion service ready to be used to download web content.

WF in Onion services.
WF attacks have been shown to be effective in identifying visits to onion services. In fact, even more so than regular websites. This is because, as described above, the onion service protocol is distinct from Tor's regular protocol: e.g., a short-lived connection to an Introduction Point precedes a long-lived connection to download content. These features were used by Kwon et al. to distinguish visits to onion services from regular websites with high accuracy. Consequently, an adversary can focus on onion services or regular websites, depending on their target. If such target is an onion service, since there are orders of magnitude fewer onion services than regular websites, identification of onion services is potentially more effective than regular websites.

Changes in Tor
The Tor Project is a very active community and the Tor software is constantly evolving. The Tor software has substantially changed since the research included in this thesis was conducted. Because of these changes, traffic generated by

new versions of the Tor software is different from the traffic collected for our experiments. However, we do not expect that all changes will impact the results of our experiments. In particular, traffic changes that are constant across websites are likely to not affect the performance of the attacks.

Below we list major changes that might make a difference in our experiments if they were reproduced on the current versions of Tor and the Tor Browser. We also argue why we believe they will or will not vary the results of such experiments.

- Onion Service protocol: since 2017, the onion service protocol has undergone a major redesign [40]. Some of the new features that *next-gen* onion services have are: a more secure directory protocol to prevent enumeration of available onion services, pinning of onion service guards, and the option to use a one-hop circuit between Rendezvous Point and the Onion service server. However, this new version of the protocol does not protect against traffic analysis attacks that allow to distinguish onion services from regular websites [41], and to fingerprinting onion services [19, 41, 55].

- Redirections to onion services: since 2018, Tor allows content providers such as Cloudflare and Facebook to use the `Alt-Svc` HTTP header to redirect visits through a special onion service proxy that they maintain [65]. This is an example of modification that is not constant across all websites and that affects fingerprinting: such redirection might introduce distinct patterns in Tor traffic that allow to distinguish pages that use such header from others.

- Netflow padding: version 0.3.1.1-alpha implements netflow padding by creating a flow of padding cells in each direction (every 1.5 to 9.5 seconds). Even though this might change Tor traffic, we do not expect a substantial variation in the performance of website fingerprinting attacks.

- Adaptive padding in Tor: since Tor version 0.4.0.5 [47], Tor implements a framework to develop padding-based countermeasures against traffic analysis. This framework allows to implement defenses such as WTF-PAD [39], among others. Since version 0.4.1.1-alpha, onion service circuits are being padded to look more similar to regular circuits and it is likely that we will see other padding strategies being implemented in the near future. These padding strategies significantly impact the results of our experiments.

2.2 Threat Model

The adversary considered in WF is a *passive* network eavesdropper. This means that they can listen to the communication but they cannot add, drop or modify networks packets. The adversary's visibility is *local* in the sense that their access is limited to a strict subset of the links in the network. In particular, in Tor the adversary is considered to sit in the last mile of the communication: they either own one or several guards or, simply, they have access to the connection between the client and the first node of the circuit (see Figure 2.5).

Entities that are in such position and thus could deploy a WF attack are multiple: local network administrators, ISPs, ASes, and other entities that are part of the Internet infrastructure. If the user is connecting wirelessly, another potential adversary is an eavesdropper who uses an antenna to intercept the communication.

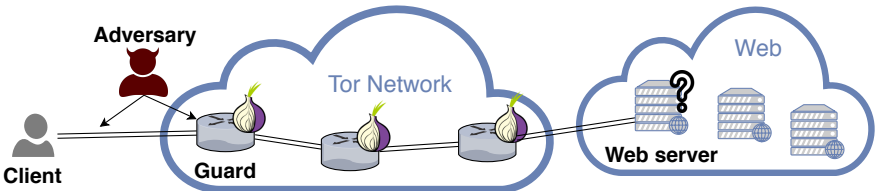


Figure 2.5: WF threat model in Tor.

2.2.1 Research assumptions

In this section we explain the most common assumptions that prior work makes on the WF's threat model.

A closed world of pages.
Early work on WF collected samples only for a small number of websites and evaluated the attack as if the user had visited one of those websites. By doing so, they implicitly assume the user will visit only pages the adversary has trained on, limiting the world of pages to a *closed world*. This is known in the literature as the closed-world assumption.

Such assumption is unrealistic as only very powerful adversaries can collect data for the whole Web [44]; but, even then, it is not known whether a classifier built with such an amount of classes – the Web size is estimated to be in the order of billions of pages [48] – would perform as well as in the closed worlds considered in the literature.

For this reason, the *closed-world* assumption has been superseded by the more realistic *open-world* assumption, in which the adversary can only train on a subset of all the existing web pages. In the open world scenario, the pages the adversary is interested in detecting are known as *monitored* pages. Conversely, pages in the complementary of the set of monitored pages are known as *unmonitored* pages.

Even though the closed world scenario is still used to benchmark defenses, most recent research includes evaluations of WF techniques in the open world.

Users only visit homepages.
Another common assumption is that users visit only website homepages. That is reflected in the data collection methodology of virtually all research on WF, as datasets used in the evaluation only include data for the homepages of the websites they consider. This is an unrealistic assumption because users do not necessarily only visit homepages. In fact, for most information look-ups on the Internet (e.g., Google searches), users visit *inner-pages*, i.e., pages in the website that are not the homepage [54].

Researchers might make this assumption because they lack the resources to crawl websites in depth. To give a sense of how the data collection would scale should they considered inner-pages, just in the English Wikipedia there are almost 6 M pages [33]. Moreover, because of the common use of templates and CMS (e.g., WordPress) – especially by the least popular websites –, websites that use the same template tend to have similar web resources. Therefore, it is likely that these pages would be more prone to be confused by the classifier, negatively affecting the performance of the attack.

It is worth mentioning that such assumption has been tested by the latest work by Panchenko et al. who captured trends by gathering URLs from popular sources such as Google search and Twitter [56]. Their study concludes that WF attacks do not scale to the size of the Web.

A page visit is clearly delimited in the traffic.
The WF adversary is often assumed to be able to collect a traffic trace for every website such that the trace solely contains traffic generated by one single visit. Equivalently, this is often phrased as an assumption on user behavior: users visit pages sequentially and they do not load the next page until the previous one has finished loading.

In practice, finding the start and end of a visit is not trivial: users may be routing other traffic over Tor or simply be browsing with multiple tabs open. At least for Firefox users, *multitab* browsing is common [51]. In Tor, traffic of different tabs is multiplexed in the same TLS connection. Therefore, the

adversary cannot use simple protocol filters to separate that traffic from traffic generated by other tabs.

Recent research shows that this assumption can be overcome by *traffic splitting* techniques [80]. This study shows how, with milder assumptions on the overlap of two consecutive page loads, the adversary can effectively detect their start and end in the traffic.

Setup replicability.
Characteristics of the client's environment such as connection properties (e.g., bandwidth and latency), geographic location, network type, processes running in the background, among others, have an effect on network traffic and, consequently, may have an impact on the accuracy of WF. Since WF researchers use the same dataset to train and test the classifier, they are assuming that the adversary is able to *replicate* the exact environment in which the attack will be deployed, i.e., the adversary collects data under the exact same conditions as the victim.

This assumption is unrealistic because, given the vast amount of variables that might differ between the adversary's and the victim's settings, it is unlikely that they will be exactly the same. Moreover, while some of these variables, such as connection properties, can be estimated at a given point in time, some others might not be observable in the traffic, e.g., background applications that do not generate network traffic.

As we will explain in Chapter 3, in our work we show the implications of such assumption for the effectiveness of the attack. Even small discrepancies between the version of the Tor Browser used by the adversary substantially degrade the performance of the attack [35].

2.3 Statistical classification

In practice, WF techniques are implemented using statistical classifiers. In this section we give a short introduction to statistical classification and explain how it is used to implement WF techniques. The summary below is based on the books "Introduction to Statistical Learning" by Hastie, Tibshirani, and Friedman [27] and "Learning Theory" by Tewari and Bartlett [73]. We strongly recommend the avid reader to expand on this summary by looking up the details in the books.

Generally speaking, the task of classification consists in arranging elements in groups, or *classes*, whose members share some characteristics. Classification is

common practice in science. For instance, biologists have defined three different *classes* – species, in biology terminology – of iris flower: setosa, virginica, and versicolor. The class of an iris flower can be determined by the length and width of its sepals and petals. However, there is high variance among flowers that belong to the same class, e.g., setosa's sepal length ranges between 4.3cm and 5.8cm, and versicolor's sepal length ranges between 4.9cm and 7cm. When we observe a new iris that falls within the intersection of these two ranges, a reasonable approach is to classify it to the class whose members are most *similar*. This, comparing an observation to existing members of a class for classification purposes, is the key intuition behind statistical classification.

Statistical classification assigns the class a new observation belongs to based on a set of examples of already-classified elements. More formally, if X is the set of examples and Y a finite set of possible classes, a statistical classifier, or simply *classifier*, is a function $f : X \to Y$ such that given an $x \in X$ for which the class is unknown, it returns a guess for x's class: i.e., $\hat{y} = f(x)$. *Classification algorithms* solve the task of finding an f that minimizes the error between $f(x)$ and the true class of x, for an arbitrary x. The space of functions that the classification algorithm searches to choose an f is called *hypothesis space*. It is common to overload the term *classifier* and call the classification algorithm, simply, *classifier*.

In machine learning, classification algorithms are part of the *supervised learning* family of algorithms, characterized by taking as input a set of observations that have been *labelled*. Popular classification algorithms are: Naïve Bayes, k-Nearest Neighbours, Support Vector Machines (SVM), Random Forests, and Artificial Neural Networks. All these classification algorithms have been used for WF. Regression is an example of supervised learning algorithm that does not solve a classification problem.

In machine learning terminology, an observation is called *instance* or *example* and is typically represented as a vector of *features*, or attributes, that describe the observation. Following the notation introduced above, we can represent the elements in X as vectors: $\vec{x} = (x_1, \ldots, x_n) \in X$, where each x_i is a feature of the instance. In Figure 2.6, we depict a set of instances for two different websites as represented in $\mathbb{R}^2$.

In order to be useful for classification, these features should take different values for different classes (high inter-class variance) and have similar values for the same class (low intra-class variance). Knowing whether a feature is relevant or not requires domain expertise.

In WF, Y is the world of web pages and X are traffic observations of those web pages, also known as *traffic traces*. Features are website characteristics based

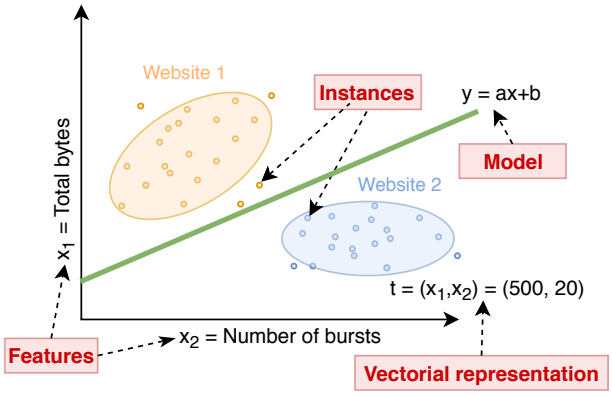


Figure 2.6: Projection of the instances of two websites over two features.

on timing and size of network packets such as the total download size in bytes or the number of bursts (see the axes in Figure 2.6).

From a statistical point of view, classification algorithms assume that there exists a joint probability distribution on the random variable $X \times Y$ that we denote by P_{XY}. Based on that assumption, observations are samples drawn from such probability distribution. Then, the classification task can be stated as: given an $x \in X$ for which we do not know its class, find y that maximizes the following probability: $P(X = x \mid Y = y)$.

In order to estimate the probabilities $P(X = x \mid Y = y)$ for all $y \in Y$, the classification algorithm uses a large sample of observations that is known as the *training set*. Classification algorithms are grouped into two main categories depending on how they estimate such probabilities: *generative*, which model the underlying probability distribution P_{XY} and use it to estimate the conditional distribution; and *discriminative*, which model the conditional distribution $P_{X|Y}$ directly.

Generative models have more descriptive power as they model how the training data was *generated* and, to classify an observation, answer the question of which class is more likely to have generated a specific observation. Since they model the distribution of how data has been generated, they can also be used to generate new data with the same distribution. However, generative models tend to be more computationally expensive than discriminative ones. Discriminative models, on the other hand, do not model how data is generated and tackle the problem in a more direct way, modeling how classes *discriminate* the data. The disadvantage is that discriminative models are more limited in capturing complex relationships in the data compared to generative ones. Whether the

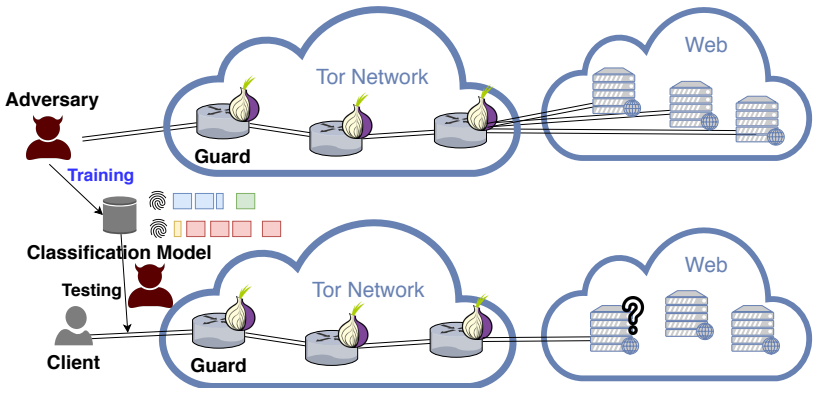


Figure 2.7: Training and testing (deployment) of a WF classifier.

classification algorithm is generative or discriminative, the resulting classifier f is often called *model* because it directly (discriminative approach) or indirectly (generative approach) models $P_{X|Y}$.

The development of a classifier can be divided into two phases: training and testing. During training, the algorithm creates the model f based on the training set. After that, the model can be used to make class predictions for a set of samples that were not in the training set, known as *test set*. Testing in this context refers to the measurement of the model's prediction error, which we explain more in detail in the next section. There are often several iterations between training and testing, in order to tune the parameters of the classification algorithm.

In WF, the adversary captures the traffic traces generated when visiting a set of pages that are likely to be visited by the user, including pages that the adversary is interested in identifying (see Figure 2.7). Next, for each page, the adversary processes the traces and extracts features that may uniquely identify it, producing a set of feature vectors. Finally, the adversary can deploy the resulting model and test traffic from an actual visit of a user, matching it to one of the pages in the dataset and thus finding out which page the user was browsing.

2.3.1 Model selection

Classifier algorithms search the hypothesis space in order to find an f that minimizes the error on class predictions. Prediction error can be decomposed into three components: error due to *bias*, error due to *variance*, and irreducible error. Error due to bias refers to error the model incurs when it is not able to capture the relationship that it is trying to model (e.g., an overly simple model).

Error due to variance occurs when the model is capturing noise that is specific to the training set and thus does not allow it to generalize well (e.g., an overly complex model). Finally, irreducible error is error that cannot be controlled by model selection such as measurement error or error intrinsic to the problem.

Bias and variance can be traded by adjusting the complexity of the model: simple models tend to reduce variance at a cost of higher bias and complex models tend to increase variance but reduce bias. When a model suffers from high variance error, we say the model is *over-fitted* and, conversely, if the model commits high bias error, we say that the model is *under-fitted*.

When evaluating a classification algorithm the machine learning engineer strives for a model that describes well the training data (low bias) but is also able to classify correctly new observations (low variance). In order to adjust the bias-variance trade-off, classification algorithms have parameters that constrain the hypothesis space, resulting in models of varying complexity. The prediction error of these models must be evaluated to inform the model selection process. However, since for most learning problems the available data is limited, a model's prediction error must be estimated statistically on samples from P_{XY}, i.e., error measurements on the sample are extrapolated to the population.

In practice, the most common statistical method to evaluate the prediction error of a model is *cross-validation*. Cross-validation is a non-parametric method, meaning that it does not require assumptions on the underlying distribution of the data. The key idea behind cross-validation is to hold out a subset of the labelled dataset for testing. The model is then trained, tuned and validated on the rest of the data and only evaluated at the end, on the data that was put aside.

In particular, k-fold cross-validation is the variant of cross-validation that is more often used to evaluate WF classifiers. In k-fold cross-validation, the dataset is divided into k disjoint sets of equal size. An estimation of the error is calculated for each of the k sets, holding such set out for testing and using the rest of the data for training. The final estimation for the prediction error is the average of the values obtained for the k folds.

The quality of the prediction error estimates vary with respect to the choice of k. For small values of k, the estimations of bias error are pessimistic because only a small part of the dataset is used for training – and we would obtain a less biased model if the whole dataset was used. In contrast, for large k, the models we train in each iteration are very similar – almost the same set of data is used to train them – increasing the variance with respect to models that are trained on other datasets. As a rule of thumb, $k = 10$ is used in practice for most purposes.

2.3.2 Prediction error

In this section we describe the metrics that are used to measure a classifier's prediction error. We begin by explaining popular metrics for binary classification and then we move to metrics for multi-class classification.

In binary classification there are only two classes: the *positive* class, the class we are interested in detecting, and the *negative* class, the complement of the positive class. Therefore, there are two possible ways in which a model can err: False Negatives (FN), positive instances that were classified as negative or False Positives (FP), negative instances that were classified as positive. Likewise, there are two types of correct predictions: correctly classified positives and negatives: True Positives (TP) and True Negatives (TN), respectively.

The most popular metrics used to measure WF classifiers in the literature are:

- *Accuracy* is the simplest metric to measure a classifier's error and is defined as the ratio of successful classifications over the total:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}\,.$$

- False Positive Rate (FPR) is the ratio of errors in the negative class:

$$\text{FPR} = \frac{FP}{FP + TN}\,.$$

- True Positive Rate (TPR), also known as *Recall*, is the ratio of correctly classified instances in the positive class:

$$\text{Recall} = \frac{TP}{TP + FN}\,.$$

- Precision is the ratio of correct classifications out of the instances that were classified as positive:

$$\text{Precision} = \frac{TP}{TP + FP}\,.$$

- The F1-Score is also used as an aggregate metric to sum up Recall and Precision in a single value. The F1-Score is defined as the harmonic mean of Recall and Precision:

$$\text{F1-Score} = 2\,\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}\,,$$

such that if either Precision or Recall are zero the F1-Score also is, and both Precision and Recall must be high in order to obtain a high F1-Score.

Another common error metric is based on the classifier's Receiver Operating Characteristic Curve (ROC curve). Most models for binary classification define a boundary between the positive and the negative class and output a confidence score based on the distance from the to-be-classified instance to such boundary. By setting a discrimination threshold on classifier confidence, one can tune the strictness of the decision boundary. In most problems there is no clear separation between the positive and the negative class and tuning the threshold allows to trade-off between TPs and FPs.

If we vary the value in the range of the discrimination threshold we obtain a curve, where each point in the curve represents the TPR-FPR trade-off of a given threshold. In Figure 2.8, we show the space for any of such curves:

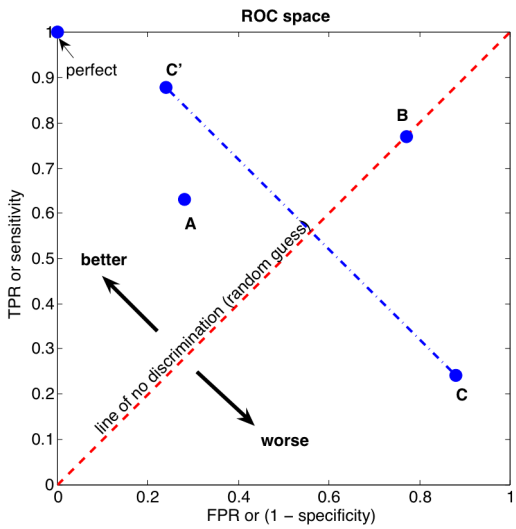


Figure 2.8: The ROC curve space. A point in the space represents a trade-off between TPR and FPR. For instance, point A is FPR=0.3 and TPR=0.65. The diagonal of the space represents random guessing (e.g., point B). The curve of the perfect classifier lies on the top-left corner. Points C and C' are mirrors over the classes: the TPR and FPR in C are the TPR and FPR in C' for the negative class. A classifier performs worse in detecting positive instances as the area under its curve on this space tends to zero. Credit: Indon, distributed under CC-BY-SA-3.0 (unmodified).

each point in the ROC space represent a pair of TPR and FPR values. A perfect classifier achieves TPR=1 and FPR=0 and hence would be represented by a curve that goes along the top-left corner. The points in the diagonal in the ROC space satisfy TPR=FPR, which is equivalently to assign positives uniformly at random between positive or negative. The area under the curve (AUC) measures the performance of the classifier: the larger the area, the less TPs the classifier has to give up to avoid FPs.

In WF, these metrics are useful in the evaluation of an open-world scenario where the monitored pages constitute the positive class. In the closed world, however, each website is a different class, hence Accuracy = Recall = Precision – because the FP errors of one class are the FNs of another and, therefore, over all the classes, FP = FN.

2.3.3 The base rate fallacy

Researchers as well as machine learning engineers might fail to accurately assess the performance that their models will have once they are deployed. These failures arise from mistakes of different nature such as mistakes in reasoning and interpretation during the evaluation process. The base rate fallacy is an example of such mistake that has been thoroughly studied in the literature on intrusion detection systems [11].

An evaluation suffers from the base rate fallacy when the base rate of the positive class, i.e., the prior probability of encountering positive instances, is overlooked or neglected. If the machine learning engineer does not take into account that the base rate, also known as *prior* probability, of the positive class is low, the measurements on prediction error are likely to be overestimated. For instance, even if a classifier exhibits a high TPR and a low FPR, there can still be orders of magnitude more FPs than TPs, since the positive class would account for a small fraction of the instances in the population, in the first place (see Figure 2.9 for a visual representation of such an imbalance).

The are multiple ways in which the base rate can be overlooked. For instance, the classification algorithm might not factor the base rate, $P(Y)$ in the estimations for $P(X = x \mid Y = y)$; the training sample might not be representative of the positive class; or, simply, the metrics used to evaluate the model measure an aspect of classifier performance that is unaffected by the base rate (e.g., Recall).

As opposed to Recall, Precision is a metric that is affected by the base rate of the positive class. Precision estimates the Bayesian Detection Rate of the classifier: i.e., the probability of a positive detected by the classifier to actually be positive. High Bayesian Detection Rate is necessary for a classifier in order

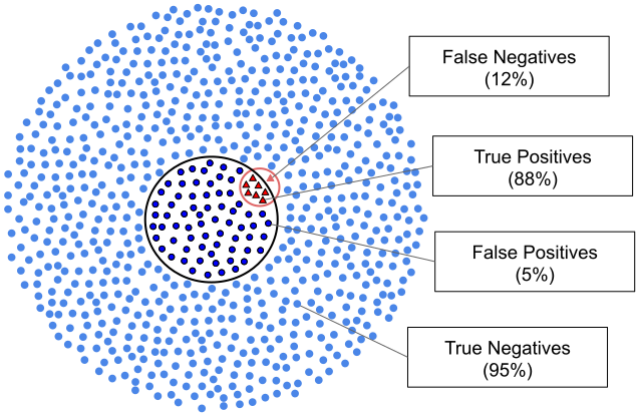


Figure 2.9: Visual representation of the base rate fallacy. Dots are elements of the negative class and triangles are elements of the positive class. Triangles within the large circle are TPs and outside of it are FNs. Dots within the large circle are FPs and outside are TNs. Even though the TPR is 88% and the FPR is 5%, Precision is only 10% (the ratio of triangles over dots within the large circle. This is because the base rate of the positive class (10%) is significantly smaller than the negative class, hence 88% of positive elements is substantially less than 5% of elements in the negative class.

to guarantee high confidence on its predictions. However, if Precision is not reported or if the sample used to measure it is biased, the evaluation might succumb to the base rate fallacy.

A common practice is to impose the same number of instances for all classes in the dataset. This is called *balancing* the dataset. The rationale for balancing the dataset is that discriminative classification algorithms (e.g., SVMs) oftentimes fail to create a model for classes that are misrepresented in the data, i.e., have a low base rate. It is important to take into account that balancing the dataset artificially modifies the base rate of the class. This might not be a problem during the training phase but, during the testing phase, balancing might overestimate the base rate for the least likely classes, hence falling into the base rate fallacy.

2.4 Website fingerprinting

In this section we describe related work on WF techniques in detail.

2.4.1 Attacks

In SSL/TLS, the adversary can usually infer the website by the IP or the domain name and perform the fingerprinting attack to identify the specific page within the site that a user visited. The world of pages in this case is orders of magnitude smaller and, therefore, the attack is much more effective. There are several papers that have explored this setting for web traffic [17,49] and we were the first in analysing it for encrypted DNS traffic [68]. In the rest of our work, however, we have focused on WF attacks against Tor, because they fundamentally undermine the privacy properties that Tor aims to provide to its users.

The first WF attack against Tor was proposed by Herrmann et al. in 2009. They used a Multinomial Naïve Bayes classifier trained on the relative frequencies of packet sizes. The evaluation was performed on a closed world of 775 websites on which the attack obtained less than 3% success rate. Moreover, the authors did not evaluate the attack in an open world.

The attack presented by Herrmann et al. was improved by Panchenko et al. a couple of years later [57]. They presented an SVM with a Radial Basis Function as kernel and refined the feature set of the attack by including several features that correlate with the traffic bursts of a web download. Traffic bursts are sudden peaks in bandwidth usage such as the ones that follow HTTP GET requests. In order to capture these bursts, Panchenko et al. proposed features based on the direction of Tor traffic (e.g., from incoming to outgoing and vice versa). They obtained 55% accuracy on the dataset that Herrmann et al. used. They were also the first to evaluate WF in an open world; in a world of 5,000 pages where 1,000 of them are monitored pages, their attack achieves 73% TPR and 0.05% FPR.

In 2012, Cai et al. presented a WF attack against Tor with a substantial increase in the attack's success rate [15]. Their attack was also based on an SVM but used a custom kernel defined by the Damerau-Levenshtein edit distance which they dubbed DLSVM. Their results show a 70% success rate on a closed world of 800 websites but they did not evaluate this attack in an open world. Wang and Goldberg notably improved the accuracy of the DLSVM by using a custom, weighted edit-distance [78] and achieved 91% success rates for the same number of websites. They also proposed several Tor-specific techniques to improve the attack effectiveness against Tor. However, these attacks are prohibitively expensive as computing the kernels for edit distances becomes computationally infeasible for large datasets. Wang and Goldberg evaluated theirs and Cai et al.'s attacks in an open world of 860 unmonitored and 40 unmonitored websites; both attacks achieved high TPR (87% for Cai et al.'s and 97% for Wang and Goldberg's) and "negligible" FPR [78].

The authors of these last two attacks then worked together to develop a new attack based on a k-NN classifier [77], hence the attack is known in the WF community as the "k-NN" attack. The features used by this classifier included thousands of features obtained from variations of simpler features that had been proposed in the literature. The classifier used a custom distance that weighs features in order to minimize the distance between instances that belong to the same website. The evaluation of the attack shows an outstanding performance: 95% success rate (closed world) on Cai et al's dataset and 85% TPR and 0.6% FPR in an open world of 5,000 websites with 100 monitored ones.

Hayes and Danezis presented k-Fingerprinting (k-FP), a novel attack based on a Random Forest and k-NN classifiers [28]. The random forest is used for a feature transformation: the leafs of the tree are a new representation of web instances that is then fed to a k-NN for the actual classification. Their open-world evaluation was the largest until then: 100,000 websites with 30 monitored ones. The results show 95% success on a 55-site closed world, and 85% TPR and 0.02% FPR in the open world.

Panchenko et al. designed a new feature set based on an SVM whose features where the cumulative sum of packet sizes [56]. They called it CUMUL and evaluated it on a realistic dataset collected from diverse sources, such as Twitter and Google search. Their closed-world evaluation was on 100 websites from Wang et al.'s dataset and obtained 91.38% on it. For the open world they used Wang et al.'s dataset and obtained 96.92% TPR and 1.98% FPR.

Finally, in recent years, deep learning techniques have been applied on WF attacks [64, 69]. These techniques have improved the success rate of the attack several percent points with respect to the attacks based on traditional machine learning techniques. They also have the advantage of not requiring feature engineering, as they automatically extract the features from the traffic traces.

A summary of the results of the attacks can be found on Table 2.1.

2.4.2 Defenses

Research on WF has taken the form of an arms race: each attack is followed by a defense that is shown to protect against it. Initial defenses were inspired by existing traffic analysis countermeasures such as traffic morphing [82]. Similar to traffic morphing, defense strategies against WF consist in delaying network packets and adding new, *dummy* packets to perturb the traffic features the attacks exploit. Adding dummy packets is commonly known as *cover traffic* or *link* padding (as opposed to packet padding, which is padding added to individual packets).

Table 2.1: The closed-world (CW) size is the number of websites in the dataset. The open-world (OW) size is the fraction of monitored sites with respect to the total. For presentation we do not include the number of instances per site in the datasets.

Attack	CW size	Acc (%)	OW size	TPR (%)	FPR (%)
Herrmann et al. [29]	775	2.96	NA	NA	NA
Panchenko et al. [57]	775	55	1,000/5,000	73	0.05
Cai et al. [15]	800	70	40/900	86.9 [78]	~ 0 [78]
OSAD [78]	800	91	40/900	96.9	~ 0
k-NN [77]	800	95	100/5,000	85	0.6
k-FP [28]	55	96	30/100,000	85	0.02
CUMUL [56]	100	91.38	100/5,000	96.92	1.98
Rimmer et al. [64]	100	96	200/400,000	80.25	9.11
Deep FP [69]	95	98.3	95/20,000	96	0.7

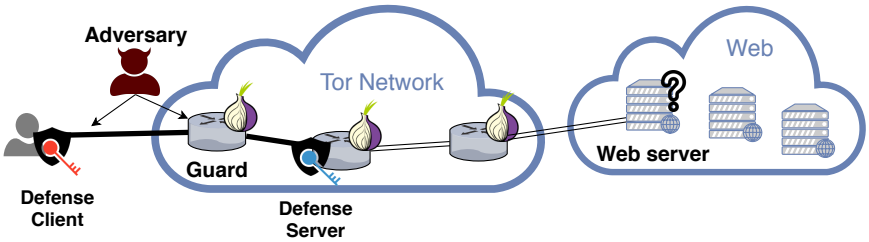


Figure 2.10: Model for network-level defenses.

Most WF defenses are designed as proxies that transform the traffic in between two Tor onion routers: the defense's client intercepts traffic coming from the client's onion router and applies the padding, and a defense server interfacing with one of the Tor relays reverses the padding transformation (see Figure 2.10). Hence, it is assumed that the defense adds a framing layer of encryption around the packets it intercepts, so that the adversary cannot pinpoint which packets are dummy.

Because these defenses are oblivious to the content of the communication, we call defenses that follow this design *network-level* defenses, as opposed to *application-level* defenses, which operate on application content. Application-level defenses are specified as running at the application layer, i.e., in the web browser or the web server. The traffic transformation is thus performed on the actual web content and it trickles down to the network traffic that the adversary observes.

The design of network-level defenses abstracts out the defense logic from the rest of the communications protocol: they have the advantage that can be plugged into Tor without modifications to the browser, the server or the onion router's source code. On the other hand, application-level defenses apply the padding at a much earlier stage, directly on the origin of the side-channel exploited by WF. For this reason, application-level defenses can be more precise in how padding is applied.

In addition, network-level defenses typically require to model the shape of application traffic as observed at the network layer, in order to generate padding that follows the same distribution. Application-level defenses eliminate this requirement, as padding is added at the application and it undergoes the same low-level transformations as legitimate application content.

Another important advantage of network-level defenses with respect to application-level ones is that they do not add bandwidth overhead on the exit node. Since the adversary's visibility is limited to the link between the client and the entry guard, padding is only required between the client and the middle node and, consequently, the defense's server could remove the padding at the middle node. Application-level padding necessarily runs end-to-end and must be removed by the server. This impacts the bandwidth overhead at the exit position. This is of paramount importance in Tor, as bandwidth in the exit position is scarce in the Tor network.

The link padding schemes that have been proposed in the literature are diverse. The following is an overview of the main categories of countermeasure schemes.

Constant-rate padding defenses.
It has long been known that padding individual packets is not sufficient to protect against WF attacks [22], as the size of a page is one of the most distinctive features and can only be concealed by link padding. Dyet et al. proposed BuFLO, a "strawman" defense that pads the communication so that the resulting stream is constant-rate, i.e., packets have the same size and the inter-packet time is fixed. The defense resulted prohibitively expensive. However, to its credit, it was not meant as a practical defense, but rather as an experiment to prove that coarse-grained features such as the size of the page leak most of the information exploited by the attacks.

Yet, the straw-man defense motivated other defenses based on constant-rate padding [13, 14], but their cost in terms of latency and bandwidth overheads discouraged their deployment in Tor. CS-Buflo and Tamaraw are the most prominent examples of constant-rate padding based defenses and both aim to minimize BuFLO's overheads. Both defenses use similar strategies to reduce bandwidth overhead: they both propose a stopping condition for padding that

allows to group websites in anonymity sets depending on their total download size; Tamaraw suggests to treat incoming and outgoing traffic independently, since incoming traffic accounts for most of the traffic in the communication; and CS-BuFLO proposes to adapt the rate in which packets are sent depending on the use of the channel. Despite high security guarantees, both Tamaraw and CS-BuFLO incur a high bandwidth and latency overhead: choosing parameters that minimize bandwidth overhead, Tamaraw and CS-BuFLO reduce attack accuracy below 50% while exhibiting 200% and 173% latency overhead, respectively. Such high latency overhead is a major drawback to consider an implementation of these defenses in Tor.

Super-sequence defenses.
Two defenses have been proposed that attempt to create anonymity sets of traffic traces [53, 77]. The key idea behind these defenses is to cluster website traffic traces by how similar they are and derive a more general trace that contains all the traces in the cluster, known as super-sequence. The padding is added to pages in the cluster so that they all look like the super-sequence trace. The main drawback for the deployment of these defenses is that they require an up-to-date database of super-sequences to be distributed to the clients which, given the fast rate in which pages change, might be expensive to maintain.

Application-level defenses.
The first and only defense implemented by Tor is an application-level defense called Randomized Pipelining (RP) [58], introduced in the previous chapter. HTTPOS is another application-level defense that is specified in the browser [45]. HTTPOS modifies the headers of HTTP requests and, similarly to RP, injects new, dummy requests strategically.

Recent work has proposed a new defense called Walkie-Talkie [80]. This defense is a hybrid between an application- and network-level defense. Walkie-Talkie requires modifications in the browser to change the communication mode to half-duplex. Half-duplex imposes that only one end can send data at a time, simplifying patterns in traffic and allowing more efficient allocation of padding. The second step of the defense mechanism is to add padding at the network layer. However, it is not specified how the client-side of the defense would coordinate with the server-side to do so.

Chapter 3

Contributions

In this chapter we contextualize the contributions of the work compiled in this thesis.

3.1 Realistic evaluation of WF techniques

The attacks presented in early WF studies were received with skepticism by the Tor community. Mike Perry, a Tor core developer strongly criticised the research assumptions made by such studies and advised to take their evaluations with a grain of salt [59]. Some of these assumptions had already been identified by Herrmann et al. [29], however, how they affected the performance of the attack in practice was an open question.

Inspired by Mike Perry's critique, we reproduced the evaluations of early WF attacks from a critical standpoint [35]. We first reviewed them and compiled a comprehensive list of assumptions on the evaluation methodology, including assumptions that were not explicitly acknowledged by the authors or that had not been identified by prior work. Then, we evaluated the impact of such assumptions on the attack performance by isolating and varying independent variables that play a role in the assumption.

For instance, to test the assumption of finding the limits of a page visit in the traffic, explained in the previous chapter, we choose the overlap of two browser tabs as the independent variable. Since the adversary cannot distinguish traffic generated by pages loaded at the same time in different tabs, they cannot deter-

mine where each page started and ended. We then measure the performance of the classifier by varying the overlap of the two tabs and fixing the rest of variables.

The variables we identified and evaluated are:

Time.
Many pages have high-frequency content changes. We show that training data *stale* over time. In our results, the accuracy of the attack decreased from 80% to less than 5% in three months. This forces the adversary to continuously train the classifier in order to keep the model up-to-date and maintain the same level of accuracy over time. In the same work, we provide an adversary model that considers the cost of periodic collection of training data.

Overlap between two tabs.
As described in Chapter 2, a common assumption made in the WF literature is that users browse pages sequentially and thus it is easy for an adversary to determine, in the traffic, when a visit to a page starts or ends. We challenge this assumption by simulating a multi-tab browsing session in our data collection process. We then apply the classifier on multi-tab data and show how the classifier breaks completely in that case.

Tor Browser version.
As a particular instance of the replicability assumption described in Chapter 2, we evaluate a scenario in which the user is using a different browser configuration to the one the adversary used to collect training data for the attack. We evaluated different versions of the Tor Browser and different values for a property in Tor's configuration. In the worst case, we observe a decrease in the performance of the attack of over 60% accuracy.

Location and Network.
Similarly, the adversary might have collected the dataset from a different network or location. We collected data from three different locations with varying network characteristics. We show that network is another variable that impacts the accuracy of the attack, to the point that, in our experiments, the performance of the attack trained on one location and tested in a different location can decrease more than 60% in accuracy.

Personalization.
Many websites personalize their content to their visitors. In those cases, the adversary might have to collect data for every personalized version of the page to maintain attack accuracy. Furthermore, some pages are behind login walls and thus the adversary is not even able to obtain training data for those. We show samples of websites that are personalized depending on the date in which

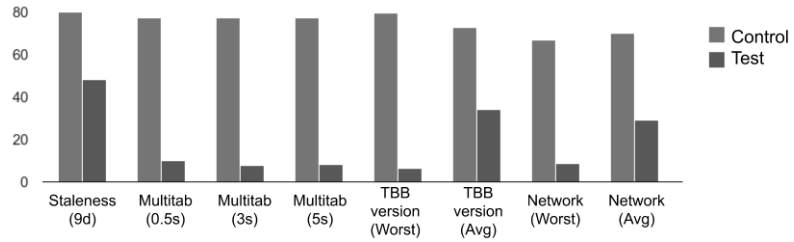


Figure 3.1: Summary of the evaluation of variables that impact attack accuracy.

the site is visited. We did not evaluate, however, the impact of such variance on attack accuracy directly.

User behavior.
Lastly, we evaluate how representative the dataset collected to train the attack is for users. Most of existing evaluations use the Alexa list of most popular sites for data collection. However, the target user might have a completely different browsing behavior and tend to visit pages that are not in Alexa. We evaluate the performance of the attack on a list of URLs collected from real users browsing the Web: the ALAD dataset. In our experiments, the classifier fails miserably when trained on data collected using the Alexa list and tested on data collected using the ALAD's list.

In Figure 3.1 we selected a representative data point for the evaluations of each of the variables enumerated above. We observe how in all the cases the accuracy of the classifier drops below 50% accuracy.

In the light of these results, we concluded that previous evaluations should be taken as an upper bound of attack effectiveness, with a caveat: although uncommon, scenarios in which the assumptions hold might exist and, in such case, the attack will be as effective as reported by the original evaluations.

Our work on this critical evaluation of attacks has steered research in the field towards answering the following question: are WF techniques deployable in practice? Panchenko et al. studied their scalability for realistic sizes of the open world and concluded that they are not; Wang and Goldberg have followed a different approach and rather aim at developing methods to mitigate the impact of the variables listed above on attack effectiveness [76, 79].

This work has also steered research in the field towards scenarios in which the adversary can scope down the classification space. Two important such scenarios are: first, Tor onion services, as Kwon et al. showed the adversary knows when the user visits an onion service, which narrows down the search only to onion

services [41]; second, HTTPS, where the adversary knows the domain name of the website and their goal is to identify the specific page within that domain.

3.1.1 The impact of the base rate fallacy

Given the difficulty of assessing the base rate of pages [66] – especially in Tor –, most prior work on WF assumes a uniform base rate, namely, all pages have the same probability of being visited. Furthermore, in many of the evaluations, classifier performance is measured with Accuracy, TPR and FPR, but Precision is not used. In our work we have thoroughly analyzed the impact of the base rate fallacy on attack performance. We have evaluated the performance of the WF classifier for a range of ratios between the monitored pages and the total size of the world and for different base rate distributions [35, 36]. We show that, for pages with a low base rate, those evaluations are biased, overestimating the performance of the classifier, which is in line with some of our critiques explained in the previous section.

In addition, we strongly recommend the use of metrics that measure classifier precision to evaluate performance. Precision is crucial in scenarios where the positive class is low, as it is the case for website fingerprinting: website popularity is a long-tail distribution with the vast majority of websites having low probability of being visited [25]. As an example of metrics that consider Precision we use F1-Score and Precision-Recall ROC curves, apart from Accuracy and traditional ROC curves [36]. We have had influence in our field since Precision has been incorporated in most recent evaluations.

3.1.2 Disparity of results not captured by averages

In most evaluations of WF attacks and defenses, researchers provide only an average for each of the metrics they use. Even though averages and aggregate metrics are useful to summarize the *overall* performance of the attack, they are not informative about the impact the attack has on individual sites. This is especially relevant for users and website developers: users have interest in knowing whether websites they visit are vulnerable or not, and website maintainers ought to know whether their website is at risk.

We have provided a breakdown of the value of popular metrics for each individual website and observe that websites are affected by the attack unevenly. In particular, the disparity is such that some websites completely evade the classifiers while others are utterly exposed by the attack. This observation set

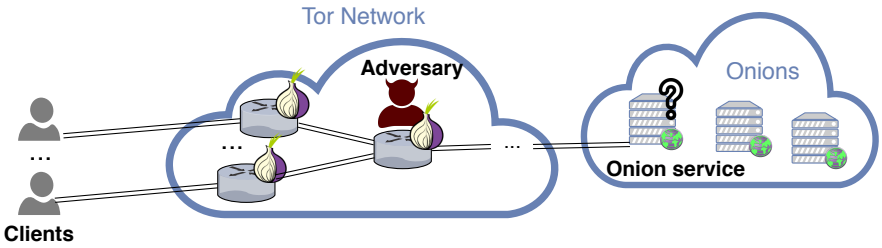


Figure 3.2: WF threat model with an adversary who controls middle nodes.

us to investigate more in depth the origin of such disparity which led to the feature analysis explained in Section 3.3.

3.2 Identification and assessment of new threats

In our studies we have explored the WF threat model by considering new variants. Our goal is to identify new attack vectors that WF enables. In this section we comment on the relevance and scope of new threat models we have studied.

3.2.1 Middle-node WF adversaries

We consider an adversary who deploys WF from middle nodes instead of guards (see Figure 3.2). The middle-node position is ideal to estimate the base rates of onion services: as opposed to the guard position, it provides more uniform samples over the user population and thus less biased samples. Such an adversary can use WF to identify visits to onion services and estimate their base rate.

Accurate estimates of the base rate of onion services can be used to adjust the base rates in the WF classifier and thus improve attack performance from the guard position. Simulating such an adversary, we measured the popularity of a well-known onion service that hosts an online social network [32].

In order to measure the base rate of the online social network we implemented a one-class classifier, i.e., the classifier only takes instances for the monitored class for training and draws a boundary around them. In Figure 3.3 we show the evaluation of the classifier in the open world and how the classification boundary (in black) was selected in order to reduce the FPR of the classifier and thus reduce the effect of the base rate fallacy. Since we do not know the base rate of the online social network a priori, in Figure 3.4 we show the performance of the

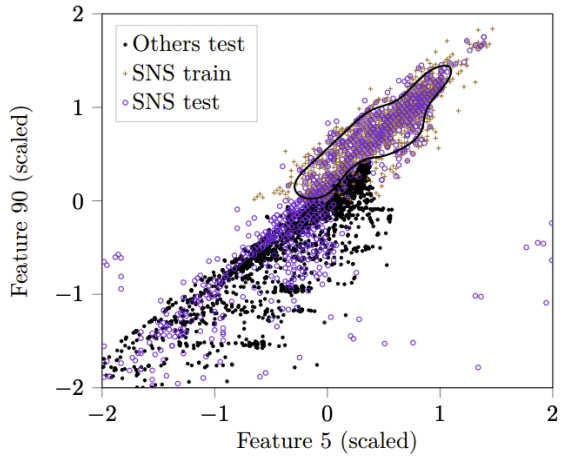


Figure 3.3: Projection over two features of the one-class classification of the onion service of an online social network. The plus sign marks are instances used for training the classifier, the circle marks are SNS instances used to test the positive class and the cross marks are instances that belong to non-SNS sites used to test the negative class. The black line shows the boundary that was learned by the classifier to reduce FPs.

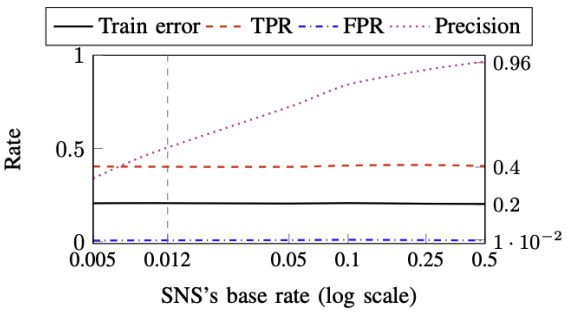


Figure 3.4: Performance of the one-class classifier for different base rates.

classifier for a range of base rates. As shown by the dashed vertical line in the figure, the classifier provides reasonable Precision for base rates as low as 0.012.

For the same reasons the middle position is ideal for measurements of onion service popularity, we argue that it provides better visibility to non-targeted

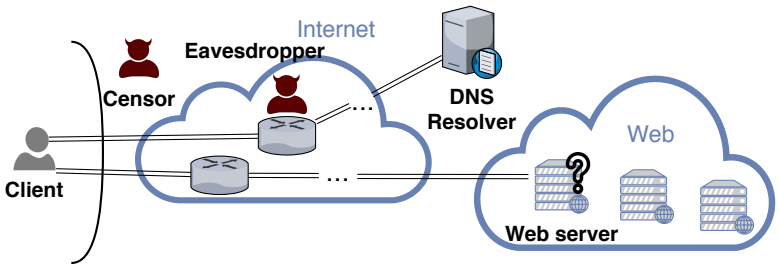


Figure 3.5: Adversary who applies WF on encrypted DNS traffic for monitoring or censoring.

attacks where the adversary might not be able to de-anonymize a user completely, but allows them to more efficiently select targets for subsequent attacks. We developed a traffic classifier that allows to identify circuit type (either onion or regular circuits), position and URL visited. This way the adversary can identify connections to websites of interest and select the guard node for more expensive or sophisticated attacks.

3.2.2 DNS-fingerprinting adversary

We have also studied threat models that fall outside the Tor's threat model. We consider an adversary who cannot see the domain because both DNS resolution and TLS's SNI are encrypted, and the IP resolves to a hosting provider or CDN that does not allow to identify the website uniquely. This scenario is becoming increasingly relevant as TLS version 1.3 supports encrypted SNI and important players in the Internet are deploying encrypted DNS solutions. In particular, Google and Cloudflare offer DNS over HTTPS (DoH) resolvers. DoH is a protocol to encrypt DNS that is also supported by major browsers such as Firefox.

In our work, we evaluate the use of WF on encrypted DNS traffic to either monitor or censor visits to websites (illustrated in Figure 3.5). We found that traditional WF features do not perform well on DNS and developed a new feature set specifically tailored for DNS traffic.

We show that it is possible to identify websites by applying WF on DoH traffic with over 90% F1-Score on a closed world of 1,500 sites. We then apply the lessons learned in our previous critical analysis to evaluate the effect of factors such as location, resolver, platform, or client on the attack's performance. Our results show that although these variables have a significant effect on attack effectiveness, they are far from completely deterring the attacks.

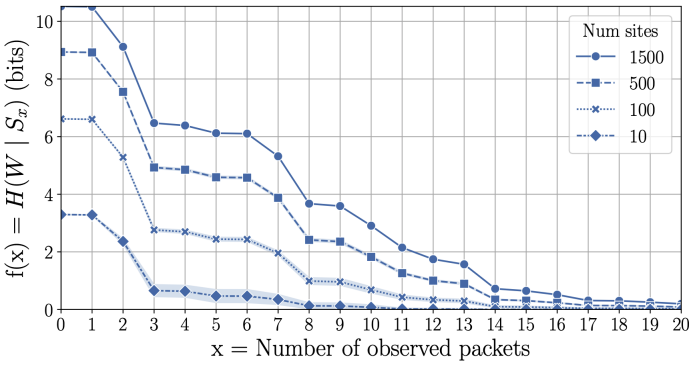


Figure 3.6: Conditional entropy given a partial observation of DoH traces (the first x number of packets) for different number of websites.

We contacted Google and Cloudflare to responsibly disclose our findings. After our interaction with Cloudflare we decided to evaluate padding countermeasures that, although partially implemented, they were not being used. The conclusion of our evaluation of countermeasures is that they do not prevent the attack: the accuracy of the attack is still high in the presence of a countermeasure. We also evaluated the use of Tor to send DoH traffic and showed that, unlike WF in regular HTTPS, Tor seems to effectively protect against WF attacks in DoH.

Finally, we also identify the cause of one of the information leaks that our attack exploits. We use an information-theoretic framework to quantify the amount of information revealed by each packet in our DoH traffic dataset. We observed a substantial amount of information revealed by the fourth packet in the DoH communication (see Figure 3.6). After further inspection, we found that the fourth packet contains the domain name of the website and its size reveals the length of such domain name. In Figure 3.7, we observe that the packet length and domain length distributions are just shifted by a constant of 51 bytes, the size of the HTTPS header. We argue that such an early leak in the communication permits the adversary to censor the subsequent communication with the web server.

We are the first to analyze the privacy of an encrypted DNS protocol. Therefore, our research raises awareness about the privacy issues that traffic analysis poses for such protocols. Moreover, this study is significant because industry is fostering the deployment and adoption of encrypted DNS.

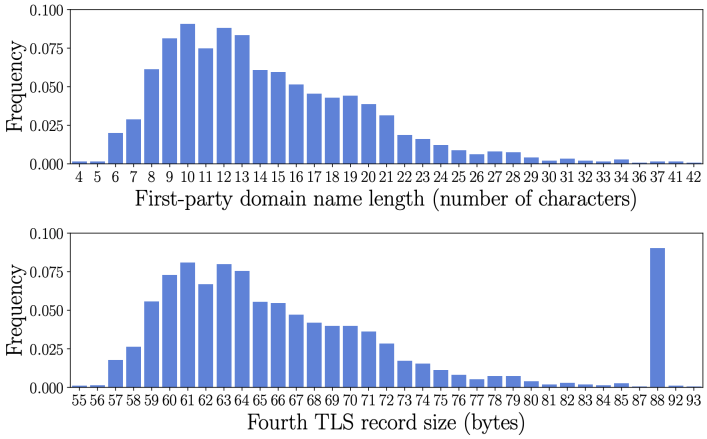


Figure 3.7: Histograms of domain name lengths (top) and fourth TLS record lengths (bottom) (normalized over the total sum of counts).

3.3 Engineering and analysis of traffic features

In 2009, Dyer et al. published an extensive study of the impact of padding-based countermeasures to WF attacks. They evaluated all existing classifiers and noticed that the choice of classification algorithm had less impact on the attack's success rate than the set of classifier features [22]. The authors evaluated different classifiers with the same set of features and did not observe a significant difference. For this reason, there is a large part of research on WF that focuses on the development of feature sets. As we described above, almost every attack uses a different set of features with more or less success in classification.

As described in the previous section, we have developed new WF features that allowed us to uncover new threats enabled by WF. In particular, we developed a novel feature set for DNS over HTTPS traffic. Such features are based on bi-grams of packet lengths which exploit the uniqueness of request-response pairs in DNS traffic [68]. We also developed features that allow to identify circuit type and circuit position from a middle node in the Tor network for the analysis of WF attacks from Tor middle nodes.

We have not only studied traditional traffic features, but also high-level features of websites. Website features are strongly correlated to network traffic features, e.g., the amount of images in a page is correlated to download size in bytes,

with the advantage that web maintainers can easily modify them by changing web server settings or simply modifying the website's layout.

We have enumerated website features and analyzed their effect on network traffic with the objective to provide design guidelines for website designers and maintainers that help them protect their sites against WF [55]. Our analysis is classifier-independent as it measures the inter- and intra-variance of features across websites, without applying a classifier. Our main conclusion is that small and dynamic websites are the most resistant to the attack. We have collaborated with engineers of the Freedom of the Press Foundation to implement such guidelines in their SecureDrop's template.

We have also used such feature analysis to obtain data for *meta-learning*: we use the results of a learner, the WF classifier, as input for another classifier whose features are website-level features. This allow us to predict vulnerability to fingerprinting without the need collect traffic data, as in traditional WF evaluations. Such meta-learner could be implemented as an online service such that website maintainers can submit the URL of their site and obtain a prediction of its "fingerprintability score".

3.4 Design and development of defenses

We have contributed to the state-of-the-art on WF defenses by proposing several novel countermeasures that we describe below.

3.4.1 WTF-PAD

We present a defense that we humorously dubbed WTF-PAD. This defense is based on adaptive padding, a traffic analysis countermeasure proposed by Shmatikov and Wang [67]. The key insight of adaptive padding is that web resources create spikes, or bursts, in traffic and that *gaps* between such bursts reveal information about the page. Adaptive padding locates statistically unlikely gaps, i.e., gaps that reveal identifying information, and covers them with padding.

This strategy is implemented by creating a histogram with the inter-packet times of a sample of the traffic to be protected. Then, on every packet that the client sends, the defense samples a time from the histogram and uses it to set a timeout. If the timeout expires, it means there is a statistically unlikely delay, and thus we must add padding.

The adaptive padding algorithm has a dual mode to add the padding. In "burst" mode, the algorithm assumes there is a burst in the communication and waits for longer periods before sending padding. In the "gap" mode, the algorithm has detected a gap in traffic and aims at add padding traffic following the timing distribution of a burst.

WTF-PAD follows a similar strategy which we have adapted to web traffic in Tor. We have extended the padding strategy to allows padding in both incoming and outgoing bursts. The defense can react to fake bursts added in one direction and simulate an HTTP request-response pattern.

In addition, we create a mechanism to tune the overhead vs protection trade-off. To do so we fit a statistical distribution to the histograms and shift the mean to skew the distribution towards shorter or longer delays, to add more or less padding, respectively.

WTF-PAD is especially suited for Tor, as its adaptive padding strategy does not add any latency overhead to the communication – it does not add any delay. Tor developers have expressed their interest in implementing WTF-PAD in Tor and an implementation of a padding framework that generalizes WTF-PAD's strategy has landed in the master branch of Tor [46]. This framework will allow to develop padding-based defenses against WF and other traffic analysis attacks such as circuit fingerprinting and end-to-end correlation attacks.

Other studies have followed probabilistic approaches similar to WTF-PAD's. The HOT research project presented a countermeasure named Adaptive Padding Early (APE) that simplifies WTF-PAD by eliminating the need to fit the probability distributions to generate the padding [62]. Moreover, new zero-latency defenses have recently been proposed following WTF-PAD's approach [26].

3.4.2 ALPaCA and LLaMA

ALPaCA and LLaMA are two application-level defenses that we have proposed to protect Tor onion services and their users against WF.

ALPaCA is, to the best of our knowledge, the first server-side WF defense that has been implemented and evaluated. ALPaCA specifies mechanisms to add padding on web objects hosted in the server: e.g., it adds a random string of a specified length in the metadata section of images or as comments to HTML and scripts. By adding padding directly to web objects, ALPaCA can define padding more efficiently than network-level defenses that pad a noisy transformation of

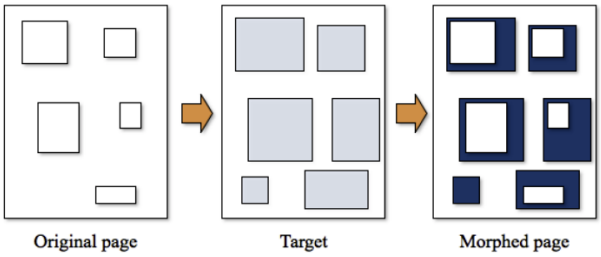


Figure 3.8: Padding applied on the resources that are embedded in the page to be protected.

object sizes observed in the network packet sizes. In Figure 3.8 we illustrate how ALPaCA adds padding to the resources embedded in the web page.

Our evaluation shows that ALPaCA reaches levels of protection that are similar to its network-level counterparts.

One of the drawbacks for the deployment of a server-side defense is the lack of incentives that general-purpose web servers have to protect against WF. We motivate ALPaCA as a defense that onion services are interested in implementing, as they particularly invested in protecting the privacy of their visitors.

The Freedom of the Press reached us and expressed interest in implementing and deployed ALPaCA in their SecureDrops. We have collaborated with one of their engineers to modify the SecureDrop page and implement a prototype of ALPaCA [1]. We expect to continue working on it and deploy the defense in the future.

In the same work we also present a client-side defense, dubbed LLaMA. LLaMA was inspired by Randomized Pipelining (RP) a WF defense that we have confirmed is not effective [35]. We have worked with the developers of RP in debugging it. We hypothesized that RP is affected by the same issues that limit the performance of HTTP pipelining: one of them is that HTTP/1.1 requires responses to be sent in-order, which can cause head-of-line blocking [60]. We hypothesized that RP is affected by the same issues that limit the performance of HTTP pipelining: one of them is that HTTP/1.1 requires responses to be sent in-order, which can cause head-of-line blocking [60]. LLaMA follows the same strategy as RP but it is implemented as a browser extension and thus does not rely on HTTP pipelining: LLaMA intercepts HTTP request using the browser's API and adds delays to simulate the shuffling performed by RP. However, the

performance of LLaMA shows marginal improvement over RP [19], indicating that the RP strategy might be inherently limited by the use of HTTP/1.1.

Despite its initial failure, LLaMA and RP form an interesting family of countermeasures (i.e., browser-level), that deserves more research. We are aware that other research groups are working on variants of LLaMA's strategy to mitigate WF attacks.

Chapter 4

Conclusion and Future Work

4.1 Conclusion

The work in this thesis has contributed to systematize the evaluation of website fingerprinting techniques. By rigorously evaluating the methodology in prior work, our research has identified flaws in previous WF models, such as unrealistic assumptions and biases in the measurement of WF classifiers [35, 36, 55]. This has been crucial to inform the development of countermeasures that mitigate WF attacks.

Moreover, we have explored the attack surface created by WF, unveiling new threats in Tor, namely: using WF from Tor middle nodes to estimate the base rate of onion services which is sensitive information and can enhance the effectiveness of the attack from the guard position; and deploying WF techniques on encrypted DNS and HTTP traffic for monitoring and blocking website visits.

In the light of these findings, we have turned to the development of countermeasures. We have proposed WTF-PAD, a lightweight defense specifically designed for Tor that does not add any delay and adds moderate bandwidth overheads to the communication. WTF-PAD's zero-latency property makes this defense specially suited for low-latency systems such as Tor. Even though such defense provides slightly lower security than its high-latency counterparts, our results show it provides sufficient protection in realistic scenarios. The analysis of this defense has contributed to the implementation of a general framework to develop traffic analysis countermeasures in Tor [47].

We identified scenarios where WF might be a significant threat. For instance, in most of our work we have focused on Tor onion services, where the world is orders of magnitude smaller than the regular Web and thus potentially easier to fingerprint. Our research has shown that the impact of WF on onion services is unevenly distributed: while average onion services tend to be more dynamic and have less resources and thus are more resistant to WF, we found a number of onion services that are critically vulnerable to WF [55].

We developed a methodology to identify the features that provide the most information to the adversary, and mapped them to features the web developers can alter via modifications on web server configuration and website's layout. We have collaborated with Freedom of the Press Foundation engineers to implement some of these modifications in the template of their SecureDrop onion services. In addition, we have developed a classifier based on such mapping between high- and low-level features that provides a user-friendly service to diagnose the exposure of websites to the attack.

Finally, inspired by the website-level protection guidelines described above, we have explored the space for defenses that could automatically apply such protections at the application layer. This has resulted in two application-level defenses, one at the client-side and the other at the server side. The latter is the first server-side defense that has been implemented against WF. We have collaborated with the Freedom of the Press Foundation engineers to implement a prototype to be deployed in their SecureDrop onion services [1].

WF is a significant privacy problem that deserves the attention of the research community; scientific articles on WF continue to appear in most top computer security conferences every year. We have improved the methodology to evaluate WF techniques and proposed effective countermeasures to mitigate WF attacks, however, there still are a number of open questions that we hope this work will encourage to answer. In the following section we enumerate such open questions.

4.2 Future work

Below we list avenues for future work that follow from our research.

WF in realistic scenarios.
Our criticisms on the assumptions made on WF studies expose a research gap: we point out that current attacks only work under laboratory conditions, but, by no means, claim that attacks that work outside the laboratory do not exist. As a matter of fact, there already is follow-up work on improving the practicality of WF attacks [76, 79] and progress on this has already allowed to relax some

of the assumptions we had identified and assessed. For instance, Wang and Goldberg have developed methods that allow to parse the traffic generated by individual website visits from the bulk of traffic [79] and Wang has proposed techniques to reduce the FPR in the open world [76]. However, there are still assumptions that remain and that future work could try to tackle. Such work may prove that accurate attacks are feasible in practice.

Provable privacy guarantees for WF defenses.
Even though there has been progress in the theoretical evaluation of WF defenses [19, 42], in practice, the evaluation of defenses is still conducted by bench-marking them using state-of-the-art attacks. This approach only provides an upper bound for the effectiveness of the defense, as new and more effective attacks may appear in the future. Future research should systematize the evaluation of WF defenses such that the privacy guarantees they provide can be proven, independently of current instantiations of the attack.

The end of the WF arms race.
The arms race between attacks and defenses has changed with the latest WF attacks [64,69]. Such attacks are based on deep learning in order to automatically extract relevant traffic features. Thus, the attacks can be applied on defended traffic and extract features the defenses did not hide, eliminating the need for feature engineering and, thus, giving an advantage to the adversary in the arms race. Future work should explore defenses based on deep learning. An interesting idea is to use Generative Adversarial Networks to automatically pit deep learning-based attacks and defenses. Such approach could shed light on the nature of the arms race, allowing to determine whether the attacker or the defender has a head start by the nature of the problem.

Interaction between WF and other traffic analysis attacks.
WF defenses can create collateral damage by exposing users to other attacks. For instance, it is important to note the various transformation in traffic of the defenses that we have proposed. Future research should investigate whether WF defenses introduce distinctive patterns that could aid other traffic analysis attacks such as end-to-end correlation attacks, described in Chapter 1 of this thesis. Should that be the case, the natural next step would then be to try to hide those patterns while preserving the defense's effectiveness against WF. This is part of the more general problem of identifying distinctive patterns in protocol specifications that could be exploited by traffic analysis [31] or even enable covert-channels [20]. A more ambitious pursuit would be to develop methods to systematically detect such patterns in protocol specifications and develop strategies to hide them.

Bibliography

[1] ALPaCA branch of SecureDrop. https://github.com/freedomofpress/securedrop/tree/alpaca.

[2] GCHQ taps fibre-optic cables for secret access to world's communications. http://www.theguardian.com/uk/2013/jun/21/gchq-cables-secret-world-communications-nsa.

[3] How the NSA is still harvesting your online data. https://www.theguardian.com/world/2013/jun/27/nsa-online-metadata-collection.

[4] OpenWPM. https://github.com/mozilla/OpenWPM.

[5] SecureDrop: the open-source whistleblower submission system. https://securedrop.org.

[6] Website Fingerprinting Defenses for Tor Hidden Services. https://www.esat.kuleuven.be/cosic/website-fingerprinting-defenses-for-tor-hidden-services/.

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

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

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

[10] Sadia Afroz, Huilin Chen, M. Javed, Marc Juarez, Vern Paxson, S. A. Qazi, S. Sajid, and Michael C. Tschantz. Transparency into the causes of website inaccessibility. In *Open Day for Privacy, Transparency and Decentralization Workshop*, 2018.

[11] Stefan Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security (TISSEC)*, 3(3):186–205, 2000.

[12] Steven M Bellovin. A technique for counting NATted hosts. In *ACM SIGCOMM Workshop on Internet measurment*, pages 267–272. ACM, 2002.

[13] Xiang Cai, Rishab Nithyanand, and Rob Johnson. CS-BuFLO: A congestion sensitive website fingerprinting defense. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 121–130. ACM, 2014.

[14] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *ACM Conference on Computer and Communications Security (CCS)*, pages 227–238. ACM, 2014.

[15] 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.

[16] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[17] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *IEEE Symposium on Security and Privacy (S&P)*, pages 191–206. IEEE, 2010.

[18] 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`.

[19] Giovanni Cherubin, Jamie Hayes, and Marc Juarez. "Website fingerprinting defenses at the application layer". In *Proceedings on Privacy Enhancing Technologies (PoETS)*, pages 168–185. De Gruyter, 2017.

[20] Roger Dingledine. One cell is enough to break Tor's anonymity. Tor Project Blog. `https://blog.torproject.org/one-cell-enough-break-tors-anonymity`, 2009. (accessed: March 10, 2019).

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

[22] 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 *IEEE Symposium on Security and Privacy (S&P)*, pages 332–346. IEEE, 2012.

[23] Tariq Elahi, Kevin Bauer, Mashael AlSabah, Roger Dingledine, and Ian Goldberg. Changing of the guards: A framework for understanding and improving entry guard selection in tor. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2012)*. ACM, October 2012.

[24] Rafael Gálvez, Marc Juarez, and Claudia Diaz. Profiling tor users with unsupervised learning techniques. In *INFER 2016*, page 16, 2016.

[25] Steven Glassman. A caching relay for the world wide web. *Computer Networks and ISDN Systems*, 27(2):165–173, 1994.

[26] Jiajun Gong and Tao Wang. Poster: Zero-delay lightweight defenses against website fingerprinting.

[27] 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.

[28] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *USENIX Security Symposium*, pages 1–17. USENIX Association, 2016.

[29] 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.

[30] Andrew Hintz. Fingerprinting websites using traffic analysis. In *Privacy Enhancing Technologies Symposium (PETS)*, pages 171–178. Springer, 2003.

[31] A. Houmansadr, C. Brubaker, and V. Shmatikov. The Parrot Is Dead: Observing Unobservable Network Communications. In *IEEE Symposium on Security and Privacy (S&P)*, pages 65–79. IEEE, 2013.

[32] 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.

[33] Johnny Au and other Wikipedia contributors. Size of Wikipedia. `https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia`. (accessed: March, 2019).

[34] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries. In *ACM Conference on Computer and Communications Security (CCS)*, pages 337–348. ACM, 2013.

[35] 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.

[36] 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 (ESORICS)*, pages 27–46. Springer, 2016.

[37] Marc Juarez and Vicenç Torra. A Self-Adaptive Classification for the Dissociating Privacy Agent. In *PST2013, the eleventh annual conference on Privacy, Security and Trust*, pages 44–50, 2013.

[38] Marc Juarez and Vicenç Torra. Towards a privacy agent for information retrieval. *International Journal of Intelligent Systems*, 28(6):606–622, 2013.

[39] Marc Juarez and Vicenç Torra. Dispa: An intelligent agent for private web search. In Guillermo Navarro-Arribas and VicenÃ§ Torra, editors, *Advanced Research in Data Privacy*, volume 567 of *Studies in Computational Intelligence*, pages 389–405. Springer International Publishing, 2015.

[40] George Kadianakis. Tor's Fall Harvest: The Next Generation of Onion Services. Tor Project Blog. `https://blog.torproject.org/tors-fall-harvest-next-generation-onion-services`, 2017. (accessed: June 25, 2019).

[41] Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas. Circuit fingerprinting attacks: Passive deanonymization of tor hidden services. In *USENIX Security Symposium*, pages 287–302. USENIX Association, 2015.

[42] Shuai Li, Huajun Guo, and Nicholas Hopper. Measuring information leakage in website fingerprinting attacks and defenses. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1977–1992. ACM, 2018.

[43] 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.

[44] Lucas Mearian. CIA-backed Cleversafe announces 10-exabyte storage system. `https://www.computerworld.com/article/2500364/cia-backed-cleversafe-announces-10-exabyte-storage-system.html`. (accessed: March, 2019).

[45] Xiapu Luo, Peng Zhou, Edmond W. W. Chan, Wenke Lee, Rocky K. C. Chang, and Roberto Perdisci. HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows. In *Network & Distributed System Security Symposium (NDSS)*. IEEE Computer Society, 2011.

[46] Nick Mathewson. New Release: Tor 0.4.0.1-alpha. Tor Project Blog. `https://blog.torproject.org/new-release-tor-0401-alpha`, 2019. (accessed: March 10, 2019).

[47] Nick Matthewson. New Release: Tor 0.4.0.5. Tor Project Blog. `https://blog.torproject.org/new-release-tor-0405`, 2019. (accessed: March 10, 2019).

[48] Maurice de Kunder. The size of the World Wide Web. `https://www.worldwidewebsize.com`. (accessed: March, 2019).

[49] 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.

[50] Shailen Mistry and Bhaskaran Raman. Quantifying Traffic Analysis of Encrypted Web-Browsing. *Project paper, University of Berkeley*, 1998. Available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.10.5823&rep=rep1&type=pdf`.

[51] Mozilla Labs. Test Pilot: Tab Open/Close Study: Results. `https://testpilot.mozillalabs.com/testcases/tab-open-close/results.html#minmax`. (accessed: March 17, 2013).

[52] S.J. Murdoch and G. Danezis. Low-Cost Traffic Analysis of Tor. In *IEEE Symposium on Security and Privacy (S&P)*, pages 183–195. IEEE, 2005.

[53] Rishab Nithyanand, Xiang Cai, and Rob Johnson. Glove: A bespoke website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, WPES '14, pages 131–134. ACM, 2014.

[54] Se Eun Oh, Shuai Li, and Nicholas Hopper. Fingerprinting keywords in search queries over tor. *Proceedings on Privacy Enhancing Technologies (PoETS)*, 2017(4):251–270, 2017.

[55] 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.

[56] Andriy Panchenko, Fabian Lanze, Andreas Zinnen, Martin Henze, Jan Pennekamp, Klaus Wehrle, and Thomas Engel. Website fingerprinting at internet scale. In *Network & Distributed System Security Symposium (NDSS)*, pages 1–15. IEEE Computer Society, 2016.

[57] 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.

[58] Mike Perry. Experimental defense for website traffic fingerprinting. Tor Project Blog. `https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting`, 2011. (accessed: October 10, 2013).

[59] Mike Perry. A Critique of Website Traffic Fingerprinting Attacks. Tor project Blog. `https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks`, 2013. (accessed: December 15, 2013).

[60] Mike Perry, Gunes Acar, and Marc Juarez. personal communication.

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

[62] Tobias Pulls. Adaptive Padding Early (APE). The HOT research project blog. `https://www.cs.kau.se/pulls/hot/thebasketcase-ape/`, 2016. (accessed: March 10, 2019).

[63] Mohammad Saidur Rahman, Payap Sirinam, Nate Matthews, Kantha Girish Gangadhara, and Matthew Wright. Tik-tok: The utility of packet timing in website fingerprinting attacks. *arXiv preprint arXiv:1902.06421*, 2019.

[64] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. In *Network & Distributed System Security Symposium (NDSS)*. Internet Society, 2018.

[65] Mahrud Sayrafi. Introducing the Cloudflare Onion Service. Cloudflare Blog. https://blog.cloudflare.com/cloudflare-onion-service/, 2018. (accessed: June 25, 2019).

[66] Quirin Scheitle, Oliver Hohlfeld, Julien Gamba, Jonas Jelten, Torsten Zimmermann, Stephen D. Strowes, and Narseo Vallina-Rodriguez. A long way to the top: Significance, structure, and stability of internet top lists. In *Internet Measurement Conference*, pages 478–493. ACM, 2018.

[67] Vitaly Shmatikov and Ming-Hsiu Wang. Timing analysis in low-latency mix networks: Attacks and defenses. *European Symposium on Research in Computer Security (ESORICS)*, 2006.

[68] Sandra Siby, Marc Juarez, Claudia Diaz, Narseo Vallina-Rodriguez, and Carmela Troncoso. Does encrypted DNS imply privacy? A traffic analysis perspective. *Submitted to the USENIX Security Symposium*, 2020.

[69] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1928–1943. ACM, 2018.

[70] Dawn Xiaodong Song, David A Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on SSH. In *USENIX Security Symposium*, volume 2001, 2001.

[71] 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.

[72] Paul F Syverson, David M Goldschlag, and Michael G Reed. Anonymous connections and onion routing. In *IEEE Symposium on Security and Privacy (S&P)*, pages 44–54. IEEE, 1997.

[73] Ambuj Tewari and Peter L. Bartlett. Learning theory. In Paulo S.R. Diniz, Johan A.K. Suykens, Rama Chellappa, and Sergios Theodoridis, editors, *Academic Press Library in Signal Processing: Signal Processing Theory and Machine Learning*, volume 1 of *Academic Press Library in Signal Processing*, chapter 14, pages 775–816. Elsevier, 2014.

[74] The Tor project. Users statistics. https://metrics.torproject.org/users.html. (accessed: March, 2019).

[75] International Telecommunication Union (ITU) UN's Telecommunication Development Bureau. ICT Facts and Figures 2017. https://www.itu.int/

en/ITU-D/Statistics/Pages/facts/default.aspx, 2017. (accessed: February 26, 2017).

[76] Tao Wang. Optimizing precision for open-world website fingerprinting. *arXiv preprint arXiv:1802.05409*, 2018.

[77] 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.

[78] 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.

[79] Tao Wang and Ian Goldberg. On realistically attacking tor with website fingerprinting. In *Proceedings on Privacy Enhancing Technologies (PoETS)*, pages 21–36. De Gruyter Open, 2016.

[80] Tao Wang and Ian Goldberg. Walkie-talkie: An efficient defense against passive website fingerprinting attacks. In *USENIX Security Symposium*, pages 1375–1390. USENIX Association, 2017.

[81] Andrew M White, Austin R Matthews, Kevin Z Snow, and Fabian Monrose. Phonotactic reconstruction of encrypted voip conversations: Hookt on foniks. In *2011 IEEE Symposium on Security and Privacy*, pages 3–18. IEEE, 2011.

[82] Charles V Wright, Scott E Coull, and Fabian Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *Network & Distributed System Security Symposium (NDSS)*. IEEE Computer Society, 2009.

Part II

Publications

List of Publications

International Journals

1. Cherubin, G., Hayes, J., and Juarez, M. "Website fingerprinting defenses at the application layer". In *Proceedings on Privacy Enhancing Technologies (PoETS)* (2017), De Gruyter, pp. 168–185

    - See p. 127

2. Juarez, M., and Torra, V. Towards a privacy agent for information retrieval. *International Journal of Intelligent Systems 28*, 6 (2013), 606–622

International Conferences and Workshops with Proceedings

1. Siby, S., Juarez, M., Diaz, C., Vallina-Rodriguez, N., and Troncoso, C. Does encrypted DNS imply privacy? A traffic analysis perspective. *Submitted to the USENIX Security Symposium* (2020)

    - See p. 251

    - Under submission to USENIX'20.

2. Sirinam, P., Imani, M., Juarez, M., and Wright, M. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *ACM Conference on Computer and Communications Security (CCS)* (2018), ACM, pp. 1928–1943

3. Jansen, R., Juarez, M., Galvez, R., Elahi, T., and Diaz, C. Inside job: Applying traffic analysis to measure tor from within. In *Network & Distributed System Security Symposium (NDSS)* (2018), Internet Society

   - See p. 205

4. Rimmer, V., Preuveneers, D., Juarez, M., Van Goethem, T., and Joosen, W. Automated website fingerprinting through deep learning. In *Network & Distributed System Security Symposium (NDSS)* (2018), Internet Society

5. Overdorf, R., Juarez, M., Acar, G., Greenstadt, R., and Diaz, C. How unique is your onion? an analysis of the fingerprintability of tor onion services. In *ACM Conference on Computer and Communications Security (CCS)* (2017), ACM, pp. 2021–2036

   - See p. 165

6. Juarez, M., Imani, M., Perry, M., Diaz, C., and Wright, M. Toward an efficient website fingerprinting defense. In *European Symposium on Research in Computer Security (ESORICS)* (2016), Springer, pp. 27–46

   - See p. 101

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

   - See p. 67

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

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

10. Juarez, M., and Torra, V. A Self-Adaptive Classification for the Dissociating Privacy Agent. In *PST2013, the eleventh annual conference on Privacy, Security and Trust* (2013), pp. 44–50

Articles in Books

1. Juarez, M., and Torra, V. Dispa: An intelligent agent for private web search. In *Advanced Research in Data Privacy*, G. Navarro-Arribas and V. Torra, Eds., vol. 567 of *Studies in Computational Intelligence*. Springer International Publishing, 2015, pp. 389–405

Publication

A Critical Evaluation of Website Fingerprinting Attacks

Publication Data

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

Contributions

- Principal author.

A Critical Evaluation of Website Fingerprinting Attacks

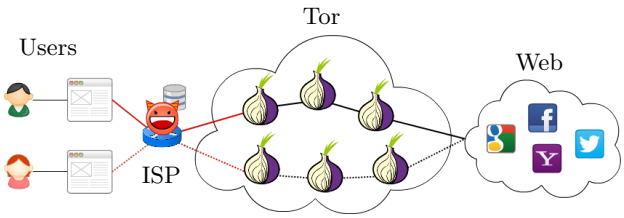
Marc Juarez[1], Sadia Afroz[2], Gunes Acar[1],
Claudia Diaz[1], and Rachel Greenstadt[3]

[1] KU Leuven, ESAT/COSIC and iMinds, Leuven, Belgium
[2] UC Berkeley, Berkeley, US
[3] Drexel University, Philadelphia, US

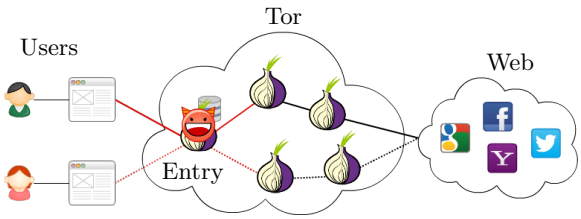
Abstract. Recent studies on Website Fingerprinting (WF) claim to have found highly effective attacks on Tor. However, these studies make assumptions about user settings, adversary capabilities, and the nature of the Web that do not necessarily hold in practical scenarios. The following study critically evaluates these assumptions by conducting the attack where the assumptions do not hold. We show that certain variables, for example, user's browsing habits, differences in location and version of Tor Browser Bundle, that are usually omitted from the current WF model have a significant impact on the efficacy of the attack. We also empirically show how prior work succumbs to the base rate fallacy in the open-world scenario. We address this problem by augmenting our classification method with a verification step. We conclude that even though this approach reduces the number of false positives over 63%, it does not completely solve the problem, which remains an open issue for WF attacks.

1 Introduction

Anonymous communication systems are designed to protect users from malicious websites and network eavesdroppers by providing means to hide the content and metadata of communications. *The Onion Router* (Tor), with about three million daily users, is the most popular anonymous communication network. It is specially designed for low-latency applications such as web browsing [8, 28]. Tor routes connections through three-hop *circuits* and encrypts the traffic in layers using *onion routing* [10], so that none of the relays can know both the origin and the destination of the communication at the same time.



(a) ISP level adversary.



(b) Malicious entry guard.

Figure 2: WF Non-targeted attacks in Tor.

Although Tor hides the routing information and communication content, the analysis of the network traffic alone may provide very rich information to an attacker with sufficient capabilities. Using timing, frequency and length of the messages, an attacker can bypass otherwise very robust security mechanisms and identify the communicating parties [7, 21].

Website fingerprinting (WF) allows an adversary to learn information about a user's web browsing activity by recognizing patterns in his traffic. The adversary in this attack compares the network traces of Tor users to a set of pre-recorded webpage fingerprints to identify the page that is being accessed. WF is different from traffic correlation attacks where the adversary has access to both entry and exit nodes and matches the patterns in the incoming and outgoing traffic to the Tor network [14]. WF is also different from deep packet inspection protocols and related traffic analysis techniques that are used to censor Tor [13].

Several previous works demonstrate the effectiveness of WF attacks on Tor despite the encryption, padding and application level defenses such as randomized pipelining [3, 11, 22, 25, 32]. Although we appreciate the importance, novelty and scientific rigor of these studies, the assumptions they made vastly simplify the problem and give unrealistic advantages to the adversary by either simplifying the world or overestimating the adversary's capabilities. Some of these assumptions are challenging and hard to attain realistically in practice [11].

For example, most current works implicitly/explicitly assume that the adversary and user both use the same Tor Browser Bundle (TBB), visit the same localized version of a limited set of pages/sites almost at the same time (or a few days apart) by using only one tab browsing. However, violating at least one of these assumptions can reduce the efficacy of the attack significantly to a point that might not make WF a threat in the real world. The authors of these studies aim to provide an upper bound for the efficacy of the attacks and argue that a particular attacker or scenario might satisfy them. Also it has been argued that in a real-world scenario, proposed countermeasures against WF would actually be more efficient than these studies have estimated [24].

The goal of this study is to assess the practical feasibility of WF attacks proposed in prior work. Our contributions and their organization in the paper are as follows:

A critical evaluation of assumptions made by prior WF studies: We provide an extensive model of the WF attack, define the assumptions made by prior WF studies on the adversary, the client-setting and the Web. We argue that these assumptions are unrealistic because they are oversimplifying the problem thus are unlikely to hold in practice (Section 3).

An analysis of the variables that affect the accuracy of WF attacks: We pin down the variables that were omitted from the models considered in previous work that have an impact on the practical effectiveness and feasibility of the attacks (Section 4). We present the results of a set of comparative experiments to evaluate the effects of these variables on traffic traces and classifier accuracy. We show that, for some of the variables, the accuracy can drop up to 70%.

An approach to reduce false positive rates: We show the effect of false positives in an open-world of 35K webpages and use *Classify-Verify* in the WF domain on the estimated probabilities of the classifier which reduces the number of false positives over 63% (Section 5).

A model of the adversary's cost: We model the cost that an adversary would incur to maintain a successful WF system (Section 6). We suspect that maintaining a perfect WF system is costly as the adversary needs to collect information about different localized versions of the webpages, user's browsing settings and update the system over time to recover from data staleness.

2 Website Fingerprinting

The main objective of an adversary in a typical WF scenario is to identify which page the user is visiting. The adversary may want to learn this information for surveillance or intelligence purposes.

The WF attack is typically treated as a classification problem, where classification categories are webpages and observations are traffic traces. The adversary first collects traffic traces by visiting webpages and trains a supervised classifier using features such as the length, direction and inter-arrival times of network packets.

Whenever a user visits a webpage over Tor, the adversary records the network trace by, for instance, intercepting the traffic locally (LAN), by having access to routers of the user's ISP, or by controlling an entry guard to the Tor network. He then runs the classifier on the intercepted network trace to guess the site the user has visited.

The first WF attacks were developed to identify pages within a single website over SSL connections [4, 19]. In 2002, Sun et al. tackled the more challenging problem of identifying individual pages within a set of websites [27] which led to Hintz's attack on an anonymizing web proxy (*SafeWeb*) [12]. Many WF studies on one-hop proxy attacks have followed [2, 9, 16, 18].

Herrmann et al. [11] deployed the first WF attack on the Tor anonymity network with only a 3% success rate for a world of 775 pages. The attacks that followed significantly improved the accuracy: Shi and Matsuura obtained 50% success rate for 20 pages [25]; Panchenko et al., obtained 54.61% accuracy using Herrmann's dataset [22]; and, finally, Cai et al. and Wang and Goldberg report success rates over 90% using edit-distance based classifiers on a world of 100 pages [3, 32].

3 Model

We model the WF adversary as *passive* and *local*: the adversary is able to eavesdrop on the user's traffic, but cannot add, drop or modify packets. We also assume that the adversary cannot decrypt the contents of the network packets, as that would render WF attacks unnecessary.

Figure 1 depicts the basic WF scenario: the attacker taps the network between the victim and the Tor entry guard and collects traffic traces, which he then compares against his database of webpage fingerprints. We make a distinction

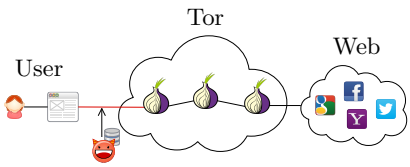


Figure 1: The basic WF targeted attack in Tor.

between two types of attacks based on the number of users targeted by the adversary and the resources at his disposal.

Targeted: In this attack, the adversary targets a specific victim to retrieve his browsing activity. This allows the attacker to train a classifier under conditions similar to those of the victim (see Figure 1), potentially increasing the success of the attack. The adversary may have enough background knowledge about the user to reproduce his configuration, or he could detect it from the observed traffic data. In Section 6, we discuss how difficult it is for the attacker to discover properties about the user's setting.

Non-targeted (dragnet surveillance): In this case, the adversary targets a set of users instead of one. ISPs, Internet exchanges and entry guard operators are in a position to deploy this attack since they can intercept the network traffic of many users (see Figures 2a and 2b, respectively). The attacker trains the classifier on a specific setting and uses the same classifier on all communications that he observes.

3.1 Assumptions

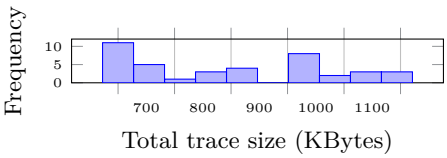
We compiled the assumptions made in the literature of WF attacks on Tor. We divided the basic model in three parts: (i) Client-side, (ii) Adversary, and (iii) Web, and classified the assumptions according to the part of the model they relate to. We note that the assumptions are not mutually exclusive and are open to other classifications. The papers that explicitly mention these assumptions are listed in Table 1.

Client-setting

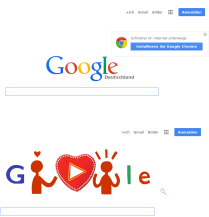
Closed-world: *There are only k webpages that the user may visit.* This is a very strong assumption because k is always very small compared to the actual number of existing webpages. Some authors have also evaluated their classifiers

in an *open-world* scenario [3, 22, 32], where there is a set of k target pages being monitored but the user is allowed to visit pages that are not in that set.

Browsing behaviour: *The users follow a specific behaviour.* For example: users browse the web sequentially, one page after the other, having only a single tab open. Nevertheless, real-world studies found that users tend to have multiple open tabs or windows [20, 30], which allow them to load several pages at once. Although we do not have access to data collected from Tor users, it is safe to think that Tor users exhibit this behavior since their connection is slower.



(a) Histogram of trace sizes



(b) Doodle and standard versions

Figure 3: Different versions of the Google homepage and corresponding histogram of trace sizes for 40 visits. The histogram shows that the distributions of packet sizes are significantly different for the same page visited on different dates.

Web

Template websites: *All websites are built using templates.* Cai et al. used a *Hidden Markov Model* (HMM) to leverage the link structure of websites for WF [3]. The authors made this assumption in order to simplify their model and reduce the number of states in the HMM.

There are further unrealistic assumptions about the Web that are not explicit but that may improve the accuracy of the WF attack. For instance, one recent study used localized (German) versions of the webpages in order to avoid different language versions [32]. In our experiments, we observed cases such as *ask.com* where the total trace size of the English version was about five times bigger than the German version of the same webpage. Given that the language of the webpage will be selected according to the Tor exit node, assuming that users would visit the same local version is a clear advantage to the adversary.

Even if we limit ourselves to localized versions of webpages, there are still other sources of dynamism such as bot detection, changing graphics and third-party

content. Figure 3a shows the histogram of sizes of 40 traces collected from *google.de* between February 14-16. We observe a clear distinction between two sets of trace sizes. The group on the left corresponds to the page without a doodle (left in Figure 3b). The group on the right with larger network footprint corresponds to the version with a special doodle for St. Valentine's day (right in Figure 3b). Note that Wang and Goldberg concluded that sites that change in size are hard to classify correctly [32].

Adversary

Page load parsing: *The adversary can detect the beginning and the end of different page loads in a traffic trace.* This has been shown to be a very hard task in the context of session reconstruction from real-world network traffic [6].

No background traffic: *The adversary can filter all background network traffic produced by other applications or other connections going through the same Tor circuit.* Tor is increasingly used in settings where multiple applications or complete operating system traffic is sent over the Tor network[4]. In these cases, separating the browsing traffic from the background traffic may present a nontrivial challenge to the adversary.

Replicability: *The adversary can train his classifier under the same conditions as the victim.* For instance, the adversary is assumed to be able to replicate client-side settings such as operating system, network connection or Tor Browser Bundle (TBB) version. This assumption allows researchers to train and test on data collected using the same settings. Depending on the type of attack this may be impossible, as the adversary may encounter difficulties in detecting and replicating users' configuration (especially in non-targeted attacks).

Table 1: Assumptions and references to papers that mention them.

Assumption	**Explicitly made by**
Closed-world	[11, 25]
Browsing behavior	[11]
Page load parsing	[3, 11, 22, 25, 32]
No background noise	[3, 11, 22, 25, 32]
Replicability	[11, 25]
Template websites	[3]

[4] `https://tails.boum.org/`

4 Evaluation

In this section we challenge some of the assumptions described in Section 3. The assumptions are: Closed-world, browsing behavior, no background traffic and replicability. For each assumption we identify the variables that are ruled out of the model by the assumption. Our objective is to measure the effect of these variables on the accuracy of WF attacks.

4.1 Datasets

We used two different lists of URLs for our crawls: the top Alexa ranking and the Active Linguistic Authentication Datatset (ALAD) [15]. The Alexa dataset is a well known list of most visited URLs that has been widely used in previous WF studies as well as in other research domains. Testing on data collected by crawling URLs in the Alexa ranking implies that the adversary always knows which pages the users are going to visit and can train a classifier on those pages. We want to test the reliability of a classifier when this assumption does not hold.

The Active Linguistic Authentication Dataset (ALAD) [15] is a dataset of web visits of real-world users, collected for the purpose of behavioral biometrics evaluation. The complete dataset contains data collected from 80 paid users in a simulated work environment. These users used the computers provided by the researchers for a total of 40 hours to perform some assigned tasks: open-ended blogging (at least 6 hours a day) and summary writing of given news articles (at least 2 hours a day). We found that these users were browsing the web to check their emails, social network sites and searching for jobs.

Table 2: ALAD dataset statistics

Top Alexa	Home page	Other pages	Not in Alexa
100	4.84%	50.34%	44.82%
1000	5.81%	60.26%	33.93%
10000	6.33%	68.01%	25.66%

The users browsed total 38,716 unique URLs (excluding news articles) which were loaded 89,719 times. These URLs include some top ranked Alexa sites, such as `google.com`, `facebook.com` and `youtube.com` and some sites that are not in Alexa top 1 million list, such as `bakery-square.com`, `miresearch.org`. Over 44% of the sites were not in the Alexa top 100, and 25% of the sites were not in the Alexa top 10000. Over 50% sites were pages other than the front

page (shown in Table 2), such as different search pages on Google/Wikipedia, a subdomain of a site (such as `dell.msn.com`) and logged-in pages of Facebook.

4.2 Data collection

To collect network traces, we used TBB combined with Selenium[5] to visit the pages, and recorded the network packets using a network packet dump tool called `dumpcap`. We used the Stem library to control and configure the Tor process [29]. Following Wang and Goldberg we extended the 10 minute circuit renewal period to 600,000 and disabled the `UseEntryGuards` to avoid using a fixed set of guard nodes [32]. We also used their methods to parse the Tor cells and remove noise by filtering acknowledgements and SENDMEs.

We crawled the webpages in *batches*. For each batch, we visited each page 4 times and collected between 5 and 10 batches of data in each crawl, resulting in 20 to 40 visits for each webpage in total. We waited 5 seconds after each page had finished loading and left 5 second pauses between each visit. The batches are collected in a round-robin fashion, hours apart from each other. We made over 50 such crawls for the experiments presented in this paper and we will share the data with other researchers upon request.

We used two physical and three cloud-based virtual machines to run crawls from different geographical locations. In order to have identical crawler configurations, we used Linux Container (LXC) based virtualization running on the same distribution and version of the GNU/Linux operating system. We disabled operating system updates to prevent background network traffic and never ran more than one crawler on a machine at the same time. We made sure that the average CPU load of the machines is low, as this may affect the WF defenses shipped in the TBB[6].

4.3 Methodology

In order to reduce the confounding effect of other variables in the measurement, we crawled the same set of webpages multiple times by changing the value of the variable under evaluation and fixing the rest of the variables. For each variable that we wanted to evaluate, we defined a *control crawl* by setting the variable to its default value (e.g., `UseEntryGuards` = 1), and a *test crawl*, by setting the variable to the value of interest (e.g., `UseEntryGuards` = 0).

[5] `http://docs.seleniumhq.org/`

[6] `https://trac.torproject.org/projects/tor/ticket/8470#comment:7`

Note that the randomized nature of the path selection algorithm of Tor and effect of time are two control variables that we cannot completely fix when measuring the effect of other variables. We tried to overcome this by using cross-validation and minimizing the time gap between the control and test crawl in all of our experiments.

These *experiments* are composed by the two following steps:

1. k-fold cross-validation using data of the control crawl.
2. Evaluate classifier's accuracy training on the control crawl and testing with data from the test crawl.

The accuracy obtained in Step 1, the case in which the adversary can train and test under the exact same conditions, is used as a baseline for comparison. We then compare the accuracy obtained in Step 2 with this baseline. We specify the details of the cross-validation in Step 1 and the testing in Step 2 later in this section.

Classifiers designed for WF attacks are based on features extracted from the length, direction and inter-arrival times of network packets, such as unique number of packet lengths or the total bandwidth consumed. The variables we evaluate in this section affect traffic features and therefore may affect each classifier in a different manner (cf., [31]). For the present study we tried to pick a classifier for each of the learning models and sets of features studied in prior work. In Table 3 we list the classifiers that we have evaluated.

Table 3: Classifiers used for the evaluation.

Name	Model	Features
`H` [11]	Naive Bayes	Packet lengths
`P` [22]	SVM	Packet lengths Order Total bytes
`D` [9]	N-grams	Total time Up/Downstream bytes Bytes in traffic bursts
`W` [32]	SVM (Fast-Levenshtein)	Cell traces
`T`	Decision tree	Same features as `P`

We observed that the relative accuracy changes are consistent across the classifiers and the variables we evaluated. In most cases, classifier `W` performed

better than the others. For this reason, our presentation of the results is focused on classifier `W`.

Classifier `W` is based on the *Fast Levenshtein-like* distance [32]. We used this classifier instead of the one based on the *OSAD*[7] distance presented in the same paper. Although the latter attained greater accuracy in Tor, it is considerably slower and can become impractical in certain circumstances, as we will further analyse in Section 6. Furthermore, for the OSAD distance we obtained over 90% accuracy scores in our control crawls and, as in the original paper, we consistently observed a 20% decrease in accuracy when evaluating classifier `W`. For this reason, we believe the results obtained with this classifier are comparable with its more accurate counterpart.

For the evaluation of each classifier we followed a similar approach to the one described by Dyer et al. First, we fixed the size of the world to a certain number of pages (k). For each page of the training crawl, we selected n_{train} random batches and picked T_{train} traffic traces in total. For each page of the testing crawl, we selected n_{test} random batches and picked T_{test} traces in total. We averaged the results by repeating each experiment m times, each time choosing a different training and testing set. Then, the accuracy of the classifier was calculated by $\frac{Total\ correct\ predictions}{Total\ test\ instances} = \frac{p}{m\,T_{test}}$, where p is the total number of correct predictions. We made sure that for the validation of the control crawl, training and testing traces were never taken from the same batch. The classification parameters are listed in Table 4.

Table 4: Description of classification parameters defined for an experiment.

Parameter	Description
k	Number of sites in the world.
$n_{train/test}$	Number of batches for training/testing.
$T_{train/test}$	Number of instances for training/testing.
p	Total number of predictions.
m	Number of trials.

From now on, we refer to *Acc control* as the average accuracy obtained for the m trials in control crawl (Step 1) and *Acc test* as the average accuracy for m trials obtained in Step 2. In the results, the standard deviation of the accuracy obtained for m trials is shown in parentheses next to the average value.

We also have designed our own attack based on decision tree learning and using the features proposed by Panchenko et al. [22]. Decision tree learning

[7] Optimal String Alignment Distance

uses binary trees as data structures to classify observations. Leaves represent class labels and nodes are conditions on feature values that divide the set of observations. Features that better divide the data according to a certain criterion (*information gain* in our case) are chosen first.

For these experiments we have extended Dyer et al.'s Peekaboo framework [9] and the source code of the classifier presented by Wang and Goldberg [32].

4.4 Time

Webpages are constantly changing their content. This is reflected in the traffic traces and consequently in the accuracy of the WF attack. In this section we used crawls of the Alexa Top 100 pages taken at different instants in time to evaluate the effect of staleness on WF.

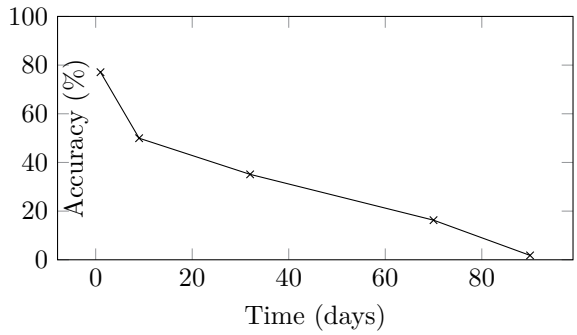


Figure 4: Staleness of our data over time. Each data point is the 10-fold crossvalidation accuracy of the classifier W which is trained using the traces from $t = 0$ and tested using traces from $0 \leq t \leq 90$. For every experiment $m = 10$, $n_{train} = 9$, $n_{test} = 1$, $T_{train} = 36$, $t_{test} = 4$ and $k = 100$.

For this experiment, we train a classifier using the traces of the control crawl, collected at time $t = 0$, and test it using traces collected within 90 days of the control crawl ($0 \leq t \leq 90$). Figure 4 shows the accuracy obtained for the W classifier for crawls over the same list of URLs at different points in time. For the rest of the classifiers we observed similar drops in accuracy. The first data point is the accuracy of the control crawl, taken at $t = 0$. The second point is taken 9 days after the control crawl ($t = 9$), and so on until the last data point that corresponds to a test crawl taken 90 days later.

We observe that the accuracy drops extremely fast over time. In our experiments it takes less than 10 days to drop under 50%. Since we observed such a drop

in accuracy due to time, we picked control and test crawls for the following experiments that fall within a period of 5 days.

This strong effect of time in the classifier's accuracy poses a challenge to the adversary who will need to train the classifier on a regular basis. Depending on the size of the world that he aims to cover, the cost of training may exceed the time in which data provides reasonable accuracy rates. We discuss this point in more detail in Section 6.

4.5 Multitab browsing

In this experiment we evaluate the success of a classifier when trained on single tab browsing, as in prior WF attacks, and tested on traces collected with multitab browsing.

In order to simulate multitab browsing behaviour, we crawled the home pages of Alexa Top 100 sites [1] while loading another webpage in the background. The background page was loaded with a delay of 0.5-5 seconds and was chosen at random from the same list, but kept constant for each batch. Then we train five classifiers from prior work, `P`, `H`, `D`, `W`, `T` (described in Table 3), using single tab traces of Alexa Top 100 webpages and test it using the multitab traces we collected. We consider a classifier successful if it can identify either the foreground page or the background page.

We observe a dramatic drop in the accuracy for all the classifiers with respect to the accuracy obtained with the control crawl (when the classifiers are trained and tested using single tab traces), even when the delay between the first page and the background page was of 0.5 seconds (Figure 5 and Table 5). We also observe a drop in the accuracy while we increase the size of the world, although the change in the accuracy was similar for all classifiers ((Figure 5).

We notice very similar accuracies for classifiers `P` and `T` in this experiment. These two classifiers are built using the same set of features but different learning models. This might imply that the specific learning model is not as important for a successful attack as the feature selection. Dyer et al. reported a similar observation between Naive Bayes and SVM classifiers.

We also vary the time gap between the two pages to account for different delays between opening the two tabs. Since the `W` classifier is based on an edit-distance, we expect the distance between the observed traffic trace and the trace of any of the two loaded pages to be smaller with respect to shorter delays, since there would be less overlap between the traffic traces of the two loaded pages. However, we do not observe a significant evidence that may support this hypothesis in

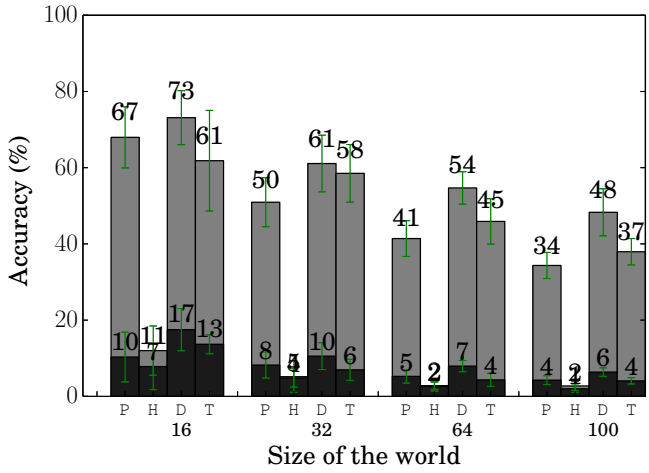


Figure 5: Average accuracies of P, H, D, T classifiers for a delay of 0.5 sec between the loading start times of the foreground page and the background page. Light gray bars represent the accuracies of the control crawls (Step 1). We plot in darker colour the accuracies obtained by training in the control and testing in the multitab crawl (Step 2). Green intervals indicate the standard deviation of the accuracy.

the evaluation for 0.5, 3 and 5 seconds of delay (Table 5) . The average page load for the test crawl for the 5 second gap experiment is 15 seconds, leaving on average 30% of the original trace uncovered by the background traffic. Even in this case, the accuracy with respect to the control crawl drops by over 68%.

Table 5: Average accuracies and standard deviations (in parentheses) of classifier W for different delays of starting the background page load. The parameters for this experiment are: $n_{train} = 9$, $n_{test} = 1$, $T_{train} = 36$, $t_{test} = 4$, $m = 10$ and $k = 100$.

Delay	Acc test	Acc control
0.5 sec	9.8% (±3.1%)	77.08% (±2.72%)
3 sec	7.9% (±0.8%)	77.08% (±2.72%)
5 sec	8.23% (±2.32%)	77.08% (±2.72%)

So far we showed the results obtained when the adversary is able to identify either the foreground page or the background page. We also consider the case

where the user utilizes a countermeasure such as Camouflage [22]. In that case, the user is not interested in the contents of the background page thus the adversary is successful only if he is able to identify the foreground page. The accuracies obtained using this definition halved the accuracies showed in Table 5 (Acc test) and Figure 5.

4.6 Tor Browser Bundle versions

In this section we evaluate the effect of different TBB versions and properties of TBB on WF. We evaluate the impact of having different TBB versions for training and testing. In practice, many TBB versions coexist, largely because of the lack of an auto-update functionality. It may be difficult for the attacker to know the exact version that is being used by the user. We also changed the configuration of Tor in the `torrc` to see how deviating from the default configuration may affect the success of the attack.

TBB Versions

We evaluate different combinations of TBB versions *2.4.7*, *3.5* and *3.5.2.1* for the control (training) and the test crawls. Table 6 shows the accuracy of classifier `W` when it trained on traces from Alexa Top 100 sites collected using TBB in the column and tested on the traces from the same sites collected using the TBB in the rows.

For versions 3.5 and 3.5.2.1 we observe high accuracies of `W` independently of the training and testing choices. This may imply that the countermeasure based on request randomization integrated in the TBB [23] may not be effective. On the other hand, when we evaluate 2.4.7 we observe low accuracies for combinations with 3.5. This is probably due to the major differences between the two versions. Version 3.5.2.1 is only a subversion of the TBB 3.5 which does not incorporate as many changes as the difference between 3.5 and 2.4.7.

TBB properties

We vary the following properties: `UseEntryGuards` and `NumEntryGuards`. `UseEntryGuards` indicates the policy for entry guard selection. It can take the following two values: *enabled*, Tor selects three entry guards for a long period of time; or *disabled*, picks one entry guard at random every time it builds a new circuit. `NumEntryGuards` sets the number of entry guards that will be available for the construction of a circuit (Default: 3). Note that even though we specify

Table 6: Entry in row X, column Y corresponds to the Acc Test (Step 2) and standard deviation (in parentheses) obtained by training in TBB version X and testing in TBB version Y. The configuration for these experiments is: $n_{train} = 9$, $n_{test} = 1$, $T_{train} = 36$, $t_{test} = 4$, $m = 10$ and $k = 100$.

TBB	2.4.7 (Test)	3.5 (Test)	3.5.2.1 (Test)
2.4.7 (Train)	62.70% (±2.8%)	29.93% (±2.54%)	12.30% (±1.47%)
3.5 (Train)	16.25% (±4.51%)	76.38% (±4.97%)	72.43% (±3.22%)
3.5.2.1 (Train)	6.51% (±1.15%)	66.75% (±3.68%)	79.58% (±2.45%)

a value for these variables, we clean the Tor data directory after each batch crawl and, therefore, entry guards possibly change across batches.

We trained and tested on the control crawl for three different pairs of values (only Step 1), listed in Table 7. The default configuration is to choose an entry guard from a list of three possible entry guards (shown in the first row of Table 7). We also evaluated the setting used by Wang and Goldberg [32], which consists in disabling `UseEntryGuards` (second row in Table 7). Finally, we enabled `UseEntryGuards` but used a list of only one possible entry guard (third row in Table 7).

Table 7: Accuracy for different entry guard configurations. For these experiments we used the following parameters: $n_{train} = 9$, $n_{test} = 1$, $T_{train} = 36$, $t_{test} = 4$, $m = 10$ and $k = 100$.

Entry guard config.	Acc control
`NumEntryGuards = 3` `UseEntryGuards = 1`	64.40% (±3.60%)
`UseEntryGuards` = 0	62.70% (±2.80%)
`NumEntryGuards` = 1 `UseEntryGuards` = 1	70.38% (±11.70%)

In Table 7 we summarize the results for these three different configurations using classifier `W`. We can see that the standard deviation increases significantly for the case where we fix one entry guard. Even though we fix the entry guard for all circuits in a batch, since we remove the Tor data directory after each batch, we force the entry guard to change. On the other hand, allowing Tor to pick a different entry guard for each circuit results in a more balanced distribution because it is more likely that the same entry guards are being used in each single batch, thus there is lower variance across batches. We must clarify that

these results are not concluding and there may be a different explanation for such difference in standard deviation.

4.7 Network

Another important variable that is being ruled out by the assumptions described in the previous section is the Internet connection. We suspect that it is unrealistic to assume the adversary is able to train using the same exact Internet connection as the user, especially in the non-targeted attack. For that, he might need more capabilities than the ones included in the basic model.

In this section we study the effect of different network locations on the accuracy of the `W` classifier. To that end, we crawled in three networks located in cities in different continents: Leuven, New York and Singapore.

Table 8: Accuracy for different network locations. The Acc test (Step 2) is calculated by training on data from *Location Train* and testing in data from *Location Test*. The parameters for the setting of these experiments are: $n_{train} = 9$, $n_{test} = 1$, $T_{train} = 36$, $t_{test} = 4$, $m = 10$ and $k = 100$.

Loc. Train	Loc. Test	Acc test	Acc control
Leuven	New York	8.83% (±2.87%)	66.95% (±2.87%)
Leuven	Singapore	9.33% (±0.98%)	66.95% (±2.87%)
Singapore	New York	68.53% (±3.24%)	76.40% (±5.99%)

Our results show that the accuracy drop between the crawls training on Leuven and testing in one of the other two locations is relatively greater than the accuracy drop observed in the experiments between Singapore and New York. Since the VM in Leuven is located within a university network and the other two VMs in data centers belonging to the same company, we attribute this difference to the fact that data center Internet connections tend to be closer to the Internet backbone. This could account for similar properties in the connection of the VMs in New York and Singapore that helped the classifier matching training and testing instances.

4.8 The importance of false positives

In this section we evaluate the open-world scenario in which an adversary monitors a small subset of the total number of pages that a user can visit, thus cannot train a classifier using every possible page.

Open-world

In the open-world scenario the adversary monitors a small number of pages and trains a classifier on traffic traces of both monitored and non-monitored pages. Following the approach described by Wang et al. [32], we assume the adversary monitors four pages: `google.com`, `facebook.com`, `wikipedia.org` and `twitter.com` and the rest of the pages in the Alexa Top 100 URLs are not monitored. We train a classifier using 36 traces for each of the Alexa Top 100 URLs, including the URLs of our monitored pages. To show the accuracy of the classifier in a true open-world scenario, we test it using four traces for each of the monitored sites plus one trace for each of the sites ranging from Alexa rank 151 to 34,710. For the classification, we assume the attacker is only interested in learning whether the user is visiting a monitored page or not, and not the exact URL that the user is visiting.

The classifier `W` offers an accuracy over 90%, a true positive rate (TPR) of 80% that is almost constant, and the false positive rate (FPR) tends to 2.6% when we increase the size of the world (Figure 6).

The base rate fallacy

Prior WF studies used accuracy-based metrics to measure the success of the WF attack in the open-world. This approach however neglects the *base rate* or *prior*, that is the probability of a user visiting a monitored page a priori. As it has been pointed out recently within the Tor community [24], this constitutes a bias in the evaluation of the WF attack called the *base rate fallacy*. Despite reporting high accuracies and low FPR when the prior is low the success of the attack can be significantly lower.

In contrast to prior work, we measure the success of the attack in the open-world using the *Bayesian detection rate* (BDR). The BDR is defined as the probability that a traffic trace actually corresponds to a monitored webpage given that the classifier recognized it as monitored.

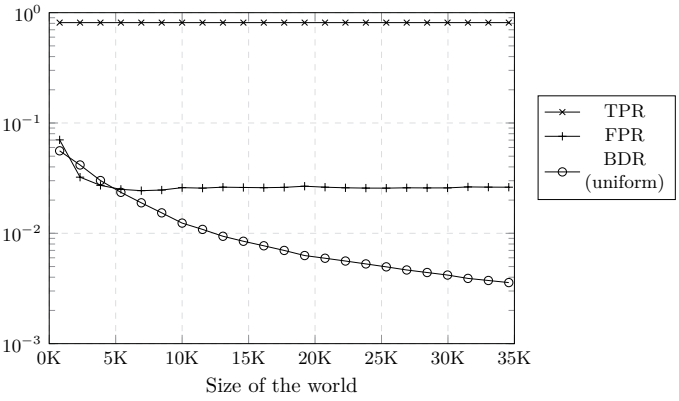


Figure 6: BDR in a uniformly distributed ∼35K open-world.

Using the Bayes theorem, the BDR is expressed as

$$P(M \mid C) = \frac{P(C \mid M)\ P(M)}{P(M)\ P(C \mid M) + P(\neg M)\ P(C \mid \neg M)}\,,$$

where M and C are the random variables of a webpage being monitored and a webpage being detected by the classifier as monitored respectively. We use the TPR as an approximation of $P(C \mid M)$ and the FPR to estimate $P(C \mid \neg M)$.

In Figure 6, we show the BDR with assuming a uniform distribution of web pages ($P(M) = \frac{|\text{Monitored}|}{|\text{World}|}$) along with the TPR and FPR of the classifier `W` for different sizes of the world. We observe that the BDR tends to zero as the size of the world increases. For a world of size 30K, which is a rather small world compared to the total size of the Web, the BDR is 0.4%. This means that there was a 0.4% probability that the classifier made a correct classification, and 99.6% of the times the adversary would wrongly conclude that the user was accessing a monitored page.

Nevertheless, assuming a uniform distribution of pages introduces a statistical bias because it underestimates the probability of visiting popular pages. In order to give a more accurate estimation of the prior we extracted statistics about visits from the ALAD dataset. In particular, we measured frequency with which the users requested the URLs of the four monitored pages and its domains. We obtained $P(M) = 0.1852$ for domains and $P(M) = 0.0352$ for homepages. We found that in our case, where the attacker only identifies home pages, the BDR tends to just 53.1% (Figure 7).

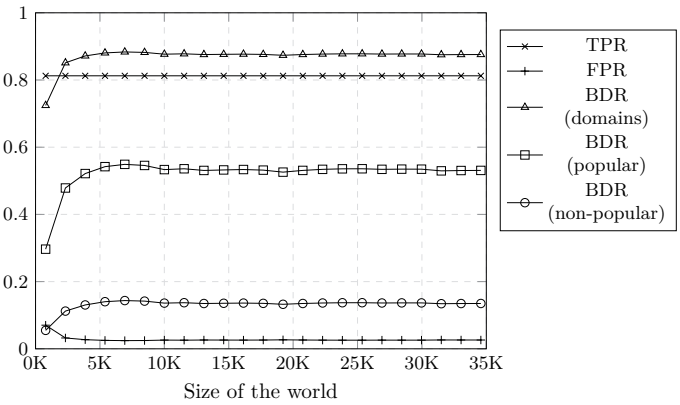


Figure 7: Evaluation of the BDR in a ∼35K open-world for multiple values of the prior $P(M)$. According to the ALAD data set, $P(M) = 0.18$ for 4 popular domains (*google.com*, *facebook.com*, *twitter.com* and *wikipedia.org*); $P(M) = 0.03$ for popular homepages and $P(M) = 0.005$ for non-popular homepages (4 random pages from ALAD with Alexa rank > 500).

We believe that a real-world adversary would not monitor homepages such as *google.com* or *facebook.com*. As a more plausible WF attack we have in mind a nation state-like adversary who is interested in identifying the access to specific pages that are difficult to block, such as an entry in a whistle-blower's blog hosted in a different country. These pages would presumably have a lower base rate than pages listed in Alexa.

To investigate the consequences of a very low prior we also calculated the BDR for monitoring a set of non-popular pages. We counted the number of visits in ALAD for 4 random pages that have a rank higher than 500 in Alexa, such as `careerbuilder.com` (rank 697) and `cracked.com` (rank 658). As expected, with a prior of 0.005 the BDR is much lower (marked with circles in Figure 7) and tending to 0.13%. Yet, note that these are just upper bounds because these monitored pages appear in the Alexa list and might be considered popular. We suspect that BDR for even more unpopular pages would be so low that would render a WF attack ineffective in this scenario.

User's browsing habits

In this section, we study the performance of a classifier that is trained on Alexa Top 100 sites and test it using real-world user visited sites. The goal is to check how successful an adversary, trained according to prior WF attacks, would be to find suspected pages on a set of pages a real-world user browsed.

We used the logs of three randomly chosen users from the ALAD and randomly picked 100 URLs from each. We crawled the URLs and collected data to feed the `W` classifier. During classification we mapped the test URLs to their top level domains. For example, `dell.msn.com` was mapped to `msn.com`. We chose to do this to not to overwhelm the classifier with false positives when it matches an inner page with the homepage. The results, summarized in Table 9 show a clear failure of the classifier in identifying the pages that these users were browsing.

Table 9: TPR and FPR for each of the users using a classifier trained on 36 traces from Alexa Top 100 sites and tested on randomly chosen 100 sites visited by ALAD User 3, 13 and 42. Here, $n_{train} = 4$, $n_{test} = 1$, $T_{train} = 36$, $t_{test} = 4$, $m = 10$ and $k = 100$.

ALAD User	TP	FP
User 3	38/260	362/400
User 13	56/356	344/400
User 42	3/208	397/400

Note that the true positive rate is the number of correct predictions over the number of predictions in which the page can be found in Alexa. The false positive rate is calculated as the number of misclassifications over the total number of predictions.

One possible reason for low TPR is due to the effect of *inner pages*. Inner pages are pages in the website that are not the homepage. We distinguish between two types of inner pages: (i) private, only accessible to the user through authentication (e.g., pages in Facebook or email accounts), and (ii), public, that is pages that are accessible for any web user but that it is not the homepage of the site. Other work has claimed that private inner pages do not matter because the TBB cleans session storage and a user has to load the login page after each session. However, we believe it is common that users leave the TBB open and visit the same inner page repeatedly within one single session. The high FPR is because the supervised classifier cannot output 'Unknown' for pages that do not exist in the training set, thus chooses to output a page in the training set that is the closest to the test page.

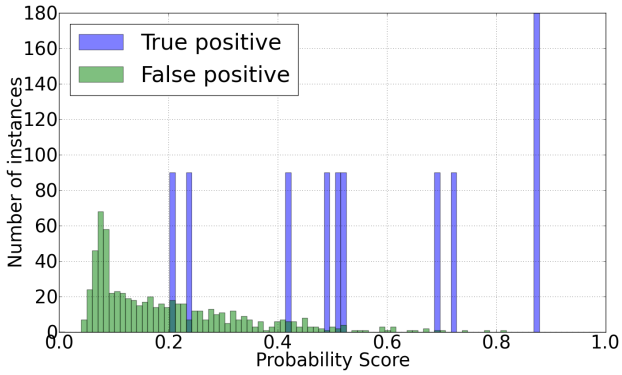


Figure 8: Estimated probability scores of the true positive and false positive instances during the open-world experiment (Section 4.8). Probability scores for the false positives are much lower than that of true positives. With a threshold ∼0.2 most of the false positives can be discarded without reducing the true positive rate.

5 Classify-Verify

In standard supervised learning, a classifier chooses at least one class even when the target class is unavailable. For example, in the open-world scenario when a classifier is trained on web pages A, B and C and tested on page D, it will choose a page from A, B and C, although the original page (D) is absent. In the open-world experiment this introduces many false positives.

During classification, a classifier can output its confidence in the classification decision in terms of posterior probability. Although standard SVM classifier (classifier `W`) does not output probabilities, an additional sigmoid function can be trained to map the SVM outputs into probabilities[8]. One way to reduce the false positive rate is to inspect the probabilities estimated by the classifier and reject the classifier's decision when the probabilities are lower than a certain threshold. This is a form of Abstaining classification [5]. In this paper, we use the modified "Classify-Verify" approach as discussed by Stolerman et al. [26].

In the open-world experiment (Section 4.8), the probability scores for true positive instances are higher than the false positive instances (Figure 8). Note that, this was a multiclass classification with 100 classes, so the random probability score of an instance is 0.01.

[8] In LibSVM, this can be achieved by simply using the `-b` option during training and testing [17].

The Classify-Verify approach adds an extra verification step after the classification. In our case, the verification process relies on a threshold that can be determined by training (Algorithm 1). When a page (D) is tested using the classifier, it outputs the probability scores of $D == A_i$ where $A_i \in \mathcal{A}$, sites in the training set. We use two verification scores based on these estimated probabilities: the maximum estimated probability, $P1$, and the difference between the maximum probability and the second highest probability, $Diff = P1 - P2$. If the verification score of D is less than the determined threshold, the classifier's output will be rejected. Unlike Stolerman et al. [26], we maximize F_β instead of $F1$ to choose threshold by adjusting weights for precision and recall. $\beta \leq 0.5$ achieves fewer false positives at the cost of true positives than $\beta > 0.5$.

Algorithm 1 Modified Classify-Verify

Input: Test page D, suspect pages $\mathcal{A} = A_1, ..A_n$ and probability scores
Output: A_D if $A_D \in \mathcal{A}$ and 'Unknown' otherwise
▷ Train a classifier
$C_\mathcal{A} \rightarrow$ classifier trained on $\mathcal{A}$
$V_\mathcal{A} \rightarrow$ verifier for $\mathcal{A}$
▷ Calculate threshold for the verifier
$t \rightarrow$ threshold maximizing F_β score
▷ Test page D
Classify D
$P_D \rightarrow$ Verification score
if $P_D >= t$ **then**
    Accept the classifier's output and return it
else
    Reject the classifier's output and return 'Unknown'
end if

5.1 Evaluation and result

We evaluate the Classify-Verify approach on the results of the open-world and ALAD experiments. To determine the threshold for a dataset, we use 10-fold cross-validation, where a threshold is determined by using 90% of the data and then tested on the remaining 10%. For F_β score, we choose $\beta = 0.5$ as we want to give more priority to precision than recall. We experimented with other β values and $F_{0.5}$ finds the best threshold (0.21 for open-world) that gives low false positives without reducing the TPR.

Our result shows that Classify-Verify reduces the number of false positives significantly without reducing the TPR. The new FPR after Classify-Verify is ~ 0.68. In the largest experiment (with $\sim$ 35K pages) the number of false positive reduces from 647 to 235, which is over 63% drop. The FPR can be reduced even further by sacrificing the TPR. The threshold estimated using $Diff = P1 - P2$ and $P1$ both perform similarly in our case.

Similarly for the users in ALAD, we determine the threshold using cross-validation. The number of false positive drops significantly (Table 10) over 50% for each of the three users.

Table 10: Classify-Verify result on the ALAD users. The number of FP drops by around 200.

ALAD User	TP	FP	New TP	New FP
User 3	38/260	362/400	31.2/260	107.6/400
User 13	56/356	344/400	26.8/356	32/400
User 42	3/208	397/400	1.0/208	41.2/400

The adversary can also use a pre-determined threshold instead of computing it every time. For example, in the open-world case if we had just chosen a threshold of 0.5 we could have discarded even more false positives with a little drop in true positives. Other more sophisticated approaches can be applied to choose a threshold, for example measuring the variance between intra- and inter-class instances. However, even after classify-verify the number false positive is lower than before but still very high. The most difficult cases are the high confidence false positives which indicate the cases where the features from censored and uncensored pages overlap.

We computed the BDR before and after applying the verification step. We used the estimation of the prior based on the prevalence of the homepages of the monitored websites in the ALAD dataset. The results show that the BDR doubles when we use the Classify-Verify approach. However, the BDR is still very low due to the exceptionally high FPR in this specific setting. For this reason, we conclude that Classify-Verify does not solve the issue completely but can be useful to partially mitigate the impact of false positives.

6 Modeling the adversary's cost

In this section, we model the cost of an adversary to maintain a WF system and discuss the scenarios in which the attack is most threatening. Current

research considers only one scenario where the adversary has the maximum information about users. Even when the adversary has all possible information, collecting, maintaining and updating these information can be costly. For example, our anecdotal experience shows that the traffic footprint of Google homepage significantly changes due to different images (doodles) embedded on the page.

A typical WF system requires 4 tasks: data collection, training, testing and updating.

Data collection cost: At first the adversary needs to collect data from the training pages. In order to maximize the classifier accuracy, the adversary may want to train with different localized versions of the same webpage and collect these under different settings, e.g., different TBB versions, user settings, entry guard configurations. If we denote the number of training pages by n, and assume that on average webpages have m versions that are different enough to reduce the classifier's accuracy, the number of pages the adversary needs to collect is $D = n \times m \times i$, where i is the number of instances per page. We denote the data collection cost as $col(D)$. This cost includes both network and storage costs.

Training Cost: In the training phase an adversary needs to train his classifier with the collected data. The training cost includes the cost of measuring features F and training a classifier C. So the cost of training the system once would be $train(D, F, C)$. If c denotes the cost of training with a single instance of a traffic trace, then the cost of training the system once would be $train(D, F, C) = D \times c$.

Testing Cost: For testing a trace, the adversary needs to collect test data T, extract features F and test using the classifier C. Let v denote the number of monitored victims and p denote the average number of pages accessed by each victim per day. Then the amount of test data is $T = v \times p$. The total test cost is $col(T) + test(T, F, C)$.

Updating Cost: To maintain the performance of the classifier, the adversary needs to update the system over time. For example the adversary might try to keep the accuracy of the classifier above a certain threshold (e.g., 50%). The updating costs include the cost of updating the data (D), measuring the features (F) and retraining the classifier (C), which is denoted as $update(D, F, C)$. If, on average, webpages change d day periods, the daily updating cost would be $\frac{update(D,F,C)}{d}$.

Then, the total cost of an adversary to maintain a WF system is:

$$init(D, F, C, T) = col(D) + train(D, F, C) + col(T) + test(T, F, C)$$

$$cost(D, F, C, T) = init(D, F, C, T) + \frac{update(D, F, C)}{d}$$

To give a more concrete example, our experiment to measure the effect of time to classifier accuracy, we found that after $d = 10$ days the accuracy attained was under 50% and thus the adversary would have needed to update his data. The $\frac{update(D,F,C)}{10}$ would not show the impossibility of a successful WF attack, but it could show that to maintain such an attack can be prohibitively expensive even for adversaries with high level of resources.

The adversary could also have extra costs before the training and the data collection. For example, the attacker could try to discover background information about the victim(s) that can be used to increase the efficiency of the attack. He could also try to discover properties about the Internet connection of the user passively. However, in perspective of the results of the previous sections, the amount of background information required to mitigate the effect of noise in the data can be much larger than previously expected. Further, a question that lingers is to what extent, given such amount of background information, the WF attack is still necessary.

7 Conclusion and future work

In this paper we studied the practical feasibility of WF attacks. We are not dismissing WF as a threat, but we suggest that more attention should be paid to the practicality of the scenarios in which these attacks are evaluated.

We studied a number of variables in isolation including website variance over time, multitab browsing behavior, TBB version, Internet connection, and the open world. When each of these assumptions are violated, the accuracy of the system drops significantly, and we have not examined in depth how the accuracy is impacted when multiple assumptions are violated.

Our results showed that success of a WF adversary depends on many factors such as temporal proximity of the training and testing traces, TBB versions used for training and testing, and users' browsing habits, which are commonly oversimplified in the WF models. Therefore, for most cases it seems that the non-targeted attack is not feasible given the sophistication level of current attacks.

There may be some exceptions in the case where a site of interest, perhaps a whistleblowing site, is particularly unique in its features and stable over time. Even the case of targeting a user is non-trivial, as these aspects of their behavior must be observed a priori or guessed correctly for WF to be a significant threat. Some users' behavior may be more susceptible to these attacks than others. In that case, the adversary could also have enough background knowledge to mount

a more targeted attack and reduce the false positive rate that we demonstrated empirically to be critical for the success of the WF adversary.

We believe that further research on evaluating the common assumptions of the WF literature is important for assessing the practicality and the efficacy of the WF attacks. Future work in developing WF attacks against Tor should also evaluate their proposed attacks in practical scenarios, so that the Tor stakeholders and the research community have a more realistic assessment of the threat they are facing.

References

[1] Alexa. Alexa Top 500 Global Site. `http://www.alexa.com/topsites`, 2014.

[2] George Dean Bissias, Marc Liberatore, David Jensen, and Brian Neil Levine. "Privacy vulnerabilities in encrypted HTTP streams". In *Privacy Enhancing Technologies Symposium (PETS)*, pages 1–11. Springer, 2006.

[3] 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.

[4] 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`.

[5] C Chow. On Optimum Recognition Error and Reject Tradeoff. *IEEE Transactions on Information Theory*, 16(1):41–46, 1970.

[6] Scott E Coull, M Patrick Collins, Charles V Wright, Fabian Monrose, Michael K Reiter, et al. On Web Browsing Privacy in Anonymized NetFlows. In *USENIX Security Symposium*, pages 339–352. USENIX Association, 2007.

[7] George Danezis. Traffic Analysis of the HTTP Protocol over TLS. *Unpublished draft*, 2009. Available at: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.3893&rep=rep1&type=pdf`.

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

[9] 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 *IEEE Symposium on Security and Privacy (S&P)*, pages 332–346. IEEE, 2012.

[10] David M Goldschlag, Michael G Reed, and Paul F Syverson. Hiding Routing Information. In *Information Hiding*, pages 137–150. Springer, 1996.

[11] 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.

[12] Andrew Hintz. Fingerprinting websites using traffic analysis. In *Privacy Enhancing Technologies Symposium (PETS)*, pages 171–178. Springer, 2003.

[13] A. Houmansadr, C. Brubaker, and V. Shmatikov. The Parrot Is Dead: Observing Unobservable Network Communications. In *IEEE Symposium on Security and Privacy (S&P)*, pages 65–79. IEEE, 2013.

[14] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries. In *ACM Conference on Computer and Communications Security (CCS)*, pages 337–348. ACM, 2013.

[15] Patrick Juola, John I Noecker Jr, Ariel Stolerman, Michael V Ryan, Patrick Brennan, and Rachel Greenstadt. A Dataset for Active Linguistic Authentication. In *IFIP WG 11.9 International Conference on Digital Forensics*. Springer, 2013.

[16] 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.

[17] LibSVM. Multi-class classification (and probability output) via error-correcting codes. `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools`, 2014.

[18] Liming Lu, EC Chang, and MC Chan. Website fingerprinting and identification using ordered feature sequences. In *European Symposium on Research in Computer Security (ESORICS)*, pages 199–214. Springer, 2010.

[19] Shailen Mistry and Bhaskaran Raman. Quantifying Traffic Analysis of Encrypted Web-Browsing. *Project paper, University of Berkeley*, 1998. Available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.10.5823&rep=rep1&type=pdf`.

[20] Mozilla Labs. Test Pilot: Tab Open/Close Study: Results. `https://testpilot.mozillalabs.com/testcases/tab-open-close/results.html#minmax`. (accessed: March 17, 2013).

[21] S.J. Murdoch and G. Danezis. Low-Cost Traffic Analysis of Tor. In *IEEE Symposium on Security and Privacy (S&P)*, pages 183–195. IEEE, 2005.

[22] 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.

[23] Mike Perry. Experimental defense for website traffic fingerprinting. Tor Project Blog. `https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting`, 2011. (accessed: October 10, 2013).

[24] Mike Perry. A Critique of Website Traffic Fingerprinting Attacks. Tor project Blog. `https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks`, 2013. (accessed: December 15, 2013).

[25] Y Shi and K Matsuura. Fingerprinting Attack on the Tor Anonymity System. In *Information and Communications Security*, pages 425–438. Springer, 2009.

[26] Ariel Stolerman, Rebekah Overdorf, Sadia Afroz, and Rachel Greenstadt. Classify, but verify: Breaking the closed-world assumption in stylometric authorship attribution. In *IFIP Working Group 11.9 on Digital Forensics*. IFIP, Springer, 2014.

[27] 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.

[28] Tor project. Users statistics. `https://metrics.torproject.org/users.html`. (accessed: March 20, 2013).

[29] Tor project. Welcome to Stem! Stem 1.1.1 Documentation. `https://stem.torproject.org`, 2014.

[30] Christian von der Weth and Manfred Hauswirth. DOBBS: Towards a Comprehensive Dataset to Study the Browsing Behavior of Online Users. *CoRR*, abs/1307.1542, 2013.

[31] T Wang and I Goldberg. Comparing Website Fingerprinting Attacks and Defenses. 2014. Technical report.

[32] 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.

A Appendices

A.1 List of used crawls

Table 11: Complete list of crawls.

Crawl Name	Date	Network	Version	Size	Batches	Accuracy control	Std
140203_042843	2/3/2014	Leuven	3.5	100	10	77.08%	± 2.72%
140203_040706	2/3/2014	Leuven	2.4.7 Alpha 1	100	10	62.70%	± 2.80%
140209_162439	2/9/2014	New York	2.4.7 Alpha 1	100	10	67.53%	± 3.91%
140214_050040	2/14/2014	Singapore	3.5	100	10	78.70%	± 4.01%
140220_042351	2/20/2014	New York	2.4.7 Alpha 1	100	10	66.05%	± 3.42%
140325_115501	3/25/2014	New York	3.5.2.1	100	10	79.58%	± 2.45%
140329_194121	3/29/2014	Singapore	3.5.2.1	100	10	76.40%	± 5.99%
140329_191630	3/29/2014	Leuven	3.5	100	10	66.95%	± 2.87%
140418_145104	4/18/2014	Leuven	3.5	100	6	54.46%	± 21.15%
140426_021609	4/26/2014	Singapore	3.5	100	10	76.93%	± 3.86%
140427_140222	4/27/2014	Leuven	3.5	100	10	71.35%	± 9.09%
140506_224307	5/7/2014	New York	3.5	100	10	77.05%	± 6.29%
140508_144031	5/8/2014	New York	3.5	100	10	72.73%	± 3.18%
140329_184252	3/29/2014	Leuven	3.5	100	10	70.38%	± 11.72%
140210_201439	2/10/2014	Leuven	2.4.7 Alpha 1	100	10	66.88%	± 5.16%
140214_040009	2/14/2014	Leuven	3.5	100	5	64.40%	± 3.60%

Publication

Toward an Efficient Website Fingerprinting Defense

Publication Data

Juarez, M., Imani, M., Perry, M., Diaz, C., and Wright, M. Toward an efficient website fingerprinting defense. In *European Symposium on Research in Computer Security (ESORICS)* (2016), Springer, pp. 27–46

Contributions

- Principal author.

Toward an Efficient Website Fingerprinting Defense

Marc Juarez[1], Mohsen Imani[2], Mike Perry[3], Claudia Diaz[1], and Matthew Wright[2]

[1] KU Leuven, ESAT/COSIC and iMinds, Leuven, Belgium,
[2] The University of Texas at Arlington, TX, USA, ,
[3] The Tor Project, https://torproject.org

Abstract. Website Fingerprinting attacks enable a passive eavesdropper to recover the user's otherwise anonymized web browsing activity by matching the observed traffic with prerecorded web traffic templates. The defenses that have been proposed to counter these attacks are impractical for deployment in real-world systems due to their high cost in terms of added delay and bandwidth overhead. Further, these defenses have been designed to counter attacks that, despite their high success rates, have been criticized for assuming unrealistic attack conditions in the evaluation setting. In this paper, we propose a novel, lightweight defense based on Adaptive Padding that provides a sufficient level of security against website fingerprinting, particularly in realistic evaluation conditions. In a closed-world setting, this defense reduces the accuracy of the state-of-the-art attack from 91% to 20%, while introducing zero latency overhead and less than 80% bandwidth overhead. In an open-world, the attack precision is just 1% and drops further as the number of sites grows.

1 Introduction

Website Fingerprinting (WF) is a type of traffic analysis attack that allows an attacker to recover the browsing history of a client. The attacker collects a database of web traffic templates and matches the client's traffic with one of the templates. WF has been shown to be effective in a wide variety of scenarios ranging from HTTPS connections [14], SSH tunnels [8], one-hop proxies [9], VPNs [18] and even anonymous communication systems such as Tor [4].

The success of WF against Tor, one of the largest deployed systems for anonymously browsing the Web [19], is particularly problematic. Tor offers stronger security than one-hop proxies and it is meant to protect against

attacks like WF that require only a local eavesdropper or a compromised guard node. However, recent WF attacks achieve more than 90% accuracy against Tor [4, 22, 23], thus breaking the anonymity properties that it aims to provide.

To counter these attacks, a broad range of defenses has been proposed. The key building block of most of these defenses is *link padding*. Link padding adds varying amounts of delays and dummy messages to the packet flows to conceal patterns in network traffic. Given that bandwidth and latency increases come at a cost to usability and deployability, these defenses must strive for a trade-off between security and performance overheads. Unfortunately, the state-of-the-art link-padding defenses are not acceptable for use in Tor: they increase latency, delaying page loads between *two* and *four* times and impose bandwidth overheads between 40% [3] and 350% [7] on average.

We note that any delays introduced by a defense are a concern for low-latency systems, as they have a direct impact on the usability of the system in interactive applications. Moderate bandwidth overheads may also impact the user experience but the load factor needs to increase substantially before being noticeable by users. Moreover, the Tor network has spare bandwidth on the ingress edge of the network, making it possible to afford a client-side defense that consumes a moderate amount of bandwidth. In this work, we thus explore the design space of effective link-padding defenses with minimal latency overhead and modest bandwidth overhead.

The contributions of the following sections are:

An analysis of the suitability of WF defenses for deployment in Tor. In Section 2, we define the threat model and give a background of existing attacks and defenses. Based on this literature review, we discuss the suitability of these defenses for an implementation in Tor.

A lightweight defense against WF attacks. We have adapted Adaptive Padding to combat WF in Tor and dubbed this new defense *Website Traffic Fingerprinting Protection with Adaptive Defense* (WTF-PAD). Section 3 gives its specification, and Section 4 presents an evaluation and a comparison of WTF-PAD with the existing WF defenses. We find that WTF-PAD is effective and has reasonable overheads for a system like Tor.

An evaluation of the defense in realistic scenarios. Prior work has shown that the accuracy of the WF attack decreases significantly when certain assumptions about the setting or user behavior do not hold [10], but to the best of our knowledge this is the first study that evaluates the effectiveness of a WF defense in these scenarios. In Section 5, we show the results for two realistic scenarios: (i) *open-world*, in which the attacker monitors a small set of web

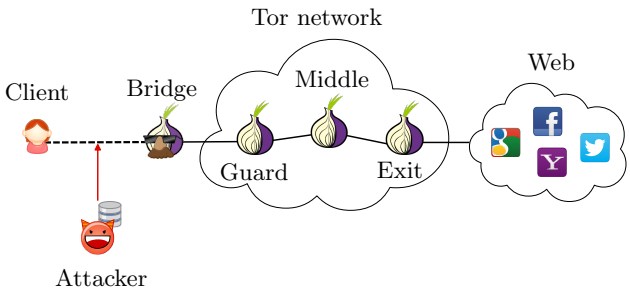


Figure 1: The WF adversary model considering Tor bridges.

pages and, (ii) *multi-tab*, where the users browse the pages using multiple tabs. We show that for these scenarios, the defense substantially reduces the accuracy of the state-of-the-art WF attack.

2 Website Fingerprinting (WF)

Tor is an overlay network that routes connections through three-hop circuits using *onion routing* [6]. The onion routers encrypt the messages in layers so that neither the relays nor localized network eavesdroppers can know both the origin and the destination of a connection.

In this paper, we assume that the client connects to Tor through a *bridge*, a volunteer-run proxy to the Tor network (see Figure 1). The adversary has access to the communication at a point between the client and the bridge. The adversary is *local*, meaning that he is unable to observe other parts of the network, and *passive*: he can observe and record packets but cannot modify, delay, drop or inject new packets. We also assume that the adversary cannot learn anything about packet payloads due to the use of layered encryption.

Defensive padding is performed end-to-end between trusted endpoints, with the adversary only having access to the padded traces. For this research, we assume the bridge is trusted. This allows to implement the defense as a *Pluggable Transport* (PT) [20], avoiding modifications in the Tor source code. Note this model is equivalent for a client connecting to the trusted entry guard without a bridge, but in that case the defense would need to be implemented at the guard.

The objective of the WF adversary is to determine what pages the user downloads over Tor by looking at the network traces. Early works on this problem [9, 18] assumed a user model that could only access a small set of pages—an assumption

that is unlikely to be met in practice. This assumption is known as the *closed-world assumption*, and it overly simplifies the problem to the point of being irrelevant to most real-world settings. In contrast, the more realistic *open-world* allows the user to visit *any* page and the attacker's goal is to determine whether the user downloads one of a small set of *monitored* pages. We have evaluated both scenarios: the closed world favors the attacker and gives a lower bound of the defense effectiveness, but our objective is to measure the performance of the defense in realistic conditions.

WF attacks are a serious threat to Tor's security: the adversary only needs the ability to eavesdrop on the client's link to the network, which can be achieved with moderate resources. With the continuous improvement in WF classifier accuracy over the past few years, this is a pressing concern. The first attack against Tor obtained 3% accuracy with a Naive Bayes classifier [8] in a closed world and without any WF countermeasures. However, the attack has been revisited with more refined feature sets [16], and state-of-the-art attacks attain over 90% accuracy [4, 15, 22, 23].

2.1 Defenses

Most of the defenses in the literature are theoretical designs without a specification for an implementation. Only a few have been evaluated for anonymous communications, and the only one that is currently implemented in Tor does not work as expected. In this section, we review WF defenses proposed in the literature and discuss their suitability for implementation in Tor.

Application-level defenses. These defenses work at the application layer. *HTTPOS* modifies HTTP headers and injects HTTP requests strategically [12], while *Randomized Pipelining*, a WF countermeasure currently implemented in the Tor Browser, randomizes the pipeline of HTTP requests. Both defenses have been shown to be ineffective in several evaluations [4, 10, 22, 23].

Supersequences and traffic morphing. Recent works have proposed defenses based on generalizing web traffic traces [2, 22]. They create anonymity sets by clustering pages and morphing them to look like the centroid of their cluster. This approach aims to optimally reduce the amount of padding needed to confound the attacker's classifier. These defenses, as well as traffic morphing techniques [11, 24], have the shortcoming that require a database of webpage templates that needs to be frequently updated and would be costly to maintain [10].

Constant-rate padding defenses. Dyer et al. evaluated the impact of padding individual packets [7], finding that this is not sufficient to hide coarse-grained features such as *bursts* in traffic or the total size and load time of the page. Dyer et al. simulated a proof-of-concept countermeasure called *BuFLO*, which used constant-rate traffic with fixed-size packets. The authors report excessive bandwidth overheads in return for moderate security. The condition to stop the padding after the transmission ends is critical to adjust the trade-off between overheads and security. BuFLO stops when a page has finished loading and a minimum amount of time has passed, not covering the size of a page that lasts longer than the minimum time.

Tamaraw [3] and *CS-BuFLO* [3, 4], both attempt to optimize the original design of BuFLO. Instead of setting a minimum duration of padding, Tamaraw stops padding when the total number of transmitted bytes is a multiple of a certain parameter. This approach groups webpages in anonymity sets, with the amount of padding generated being dependent on the webpage's total size. Given the asymmetry of web browsing traffic, Cai et al. also suggest treating incoming and outgoing traffic independently, using different packet sizes and padding at different rates. Furthermore, the authors sketched CS-BuFLO as a practical version of BuFLO, extended with congestion sensitivity and rate adaptation. Following Tamaraw's grouping in anonymity sets by page size, they propose either padding up to a power of two, or to a multiple of the amount of transmitted application data.

We question the viability of the BuFLO-based defenses for Tor. Their latency overheads are very high, such as two-to-three times longer than without defense, and the bandwidth overheads for BuFLO and CS-BuFLO are over 100%. In addition, due to the popularity of dynamic web content, it is challenging to determine when a page load completes, as needed in Tamaraw and CS-BuFLO. Nevertheless, in this paper, we compare our system against these defenses because they are the closest to meeting the deployment constraints of Tor.

3 Adaptive Padding

Adaptive Padding (AP) was proposed by Shmatikov and Wang to defend against end-to-end traffic analysis [17]. Even though WF attacks are significantly different from these end-to-end attacks, AP can be adapted to protecting against WF due to its generality and flexibility. AP has the defender examine the outgoing traffic pattern and generate dummy messages in a targeted manner to disrupt distinctive features of the patterns — "statistically unlikely" delays between packets. Shmatikov and Wang showed that with 50% bandwidth

overhead, the accuracy of end-to-end timing-based traffic analysis is significantly degraded [17].

In the BuFLO family of defenses, the inter-arrival time between packets is fixed and application data is delayed, if needed, to fit the rigid schedule of constant packet timings. This adds delays in the common case that multiple real cells are sent all at once, making this family of defenses ill-suited for a system like Tor, as it would significantly harm user experience. By contrast, Adaptive Padding (AP) does not delay application data; rather, it sends it immediately. This minimal latency overhead makes AP a good candidate for Tor.

In the rest of this section, we describe AP and explain how we adapt it to defend against WF attacks in Tor.

3.1 Design Overview

To clarify the notation adopted in this paper, we use *outgoing* to refer to the direction from the PT instance running at the client to the PT at the bridge, and conversely, *incoming* is the direction from the PT server to the client.

The basic idea of AP is to match the gaps between data packets with a distribution of generic web traffic. If an unusually large gap is found in the current stream, AP adds padding in that gap to prevent long gaps from being a distinguishing feature. Shmatikov and Wang recognized the importance of bursts in web traffic and thus developed a dual-mode algorithm. In *burst mode*, the algorithm essentially assumes there is a burst of real data and consequently waits for a longer period before sending any padding. In *gap mode*, the algorithm assumes there is a gap between bursts and consequently aims to add a fake burst of padding with short delays between packets. In this paper, we follow Shmatikov and Wang and define a burst in terms of bandwidth: a burst is a sequence of packets that has been sent in a short time period. Conversely, a gap is a sequence of packets that are spread over a long timespan.

AP algorithm. The AP algorithm is defined by two histograms of delays that we call H_B (used in burst mode) and H_G (used in gap mode). The histograms have a set of bins that spans over the range of possible inter-arrival times. Each bin contains a number of *tokens*, which can be interpreted as the probability of selecting an inter-arrival time within the range of delays represented by that bin. The last bin, which we dub the "infinity bin", includes all possible values greater than the second-to-last bin. For more details on how these histograms are defined in WTF-PAD we refer the reader to Appendix A.1.

AP implements the state machine shown in Figure 2 in each defense endpoint, i.e. both PT client and server. For simplicity, let us consider just the client's state machine in the following explanation. The operation of the server is symmetrical.

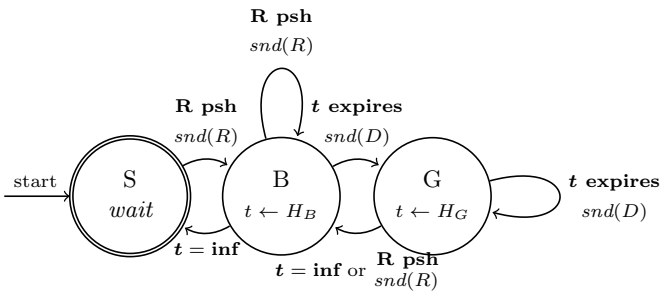


Figure 2: AP algorithm as a finite state machine as implemented in the PT client. The events are in bold and the actions in italics. The action ($snd(\cdot)$) refers to sending messages, either *real* (R) or *dummy* (D). The **psh** event means a message pushed from the application (Tor browser) to the PT client.

Burst mode. As depicted in the diagram, AP starts idle (state S) until the packet with the HTTP request is pushed from the browser (R). This causes it to enter burst mode (state B), drawing a delay t from the H_B histogram. Then it starts to count down until either new data is pushed or t expires. In the first case, the data is immediately forwarded, a new delay is sampled and the process is repeated again, i.e. it remains in burst mode. Otherwise, a dummy message (D) is sent to the other end and AP switches to state G (gap mode). The H_B histogram is built using a large dataset of web traffic, out of which we sample the times between the end of a burst and the beginning of the following burst (see Section 3.3). Therefore, while we are in a burst, the delays we sample from H_B will not expire until we find an inter-arrival time that is longer than typical within a burst, which will make the delay expire and trigger the G state.

Gap mode. While AP is in state G, it samples from histogram H_G and sends dummy messages when the times it samples expire. The histogram for gap mode, H_G, is built from a sample of inter-arrival times *within* a burst in traffic collected for a large sample of sites. That is, by sending packets with inter-arrival times drawn from H_G, we are able to generate fake bursts that follow the timing distribution of an average burst. A transition from G back to B occurs upon either sampling a token from the infinity bin or receiving a real packet. Similarly, a transition from B to S happens when we sample a token from the infinity bin.

Note that AP immediately forwards all application data. Since sending a real packet means that the timeout expired, AP has to correct the distribution by returning the token to its bin and removing a token from the bin representing the actual delay. This prevents the combined distribution of padding and real traffic from skewing towards short values and allows AP to adapt to the current transmission rate [17]. If a bin runs out of tokens, to minimize its effect on the resulting distribution of inter-arrival times, we remove tokens from the next non-empty greater bin [17]. In case all bins are empty, we refill the histogram with the initial sample.

3.2 WTF-PAD

We propose a generalization of AP called *Website Traffic Fingerprinting Protection with Adaptive Defense (WTF-PAD)*. WTF-PAD includes implementation techniques for use in Tor and a number of link-padding primitives that enable more sophisticated padding strategies than the basic AP described above. These features include:

Receive histograms. A key feature to make padding realistic is to send padding messages as a response to messages received from the other end. In WTF-PAD, we implement this by keeping another AP state machine that reacts to messages received from the other PT endpoint: the PT client has a *rcv* event when it gets a packet from the PT server. This allows us to encode dependencies between incoming and outgoing bursts and to simulate request-response HTTP transactions with the web server. Padding introduced by the *rcv* event further distorts features on bursts, as just one packet in the outgoing direction might split an incoming burst as considered by the attacks in the literature.

Control messages. WTF-PAD implements control messages to command the PT server padding from the PT client. Using control messages, the client can send the distribution of the histograms to be used by the PT server. This way, the PT client is in full control of the padding scheme. It can do accounting on received padding traffic and alert the user if relays in its circuits are sending unscheduled padding.

Beginning of transmission. Control messages can also be used to signal the beginning of the transmission. If we are in state S and a new page is requested, we will need to flag the server to start padding. Otherwise, the transmission from the first request to the following response is uncovered and reveals the size of the `index.html` page.

Soft stopping condition. In contrast to Tamaraw and CS-BuFLO, WTF-PAD does not require an explicit mechanism to conceal the total time of the transmission. At the end of the transmission, the padding is interrupted when we hit the infinity bin in the gap state and then the infinity bin in the burst state. See the Appendix A.1 for further discussion on how to set the tokens in the infinity bins. The lack of a firm stop condition represents an advantage over existing link-padding-based defenses, which require a mechanism to flag the boundaries of the transmission. The probability of stopping will depend on the shape of the histograms at the end of the transmission.

3.3 Inter-arrival time distributions

Shmatikov and Wang did not specify in the original AP paper how to build and use the distribution of inter-arrival times in the AP histograms. In their simulations, they sampled the inter-arrival times for both real and padding traffic from the same distribution. To build the histograms, we have sampled the times from a crawl of the top 35K pages in the Alexa list. First, we uniformly selected a sample of approximately 4,000 pages and studied the distribution of inter-arrival times within their traces.

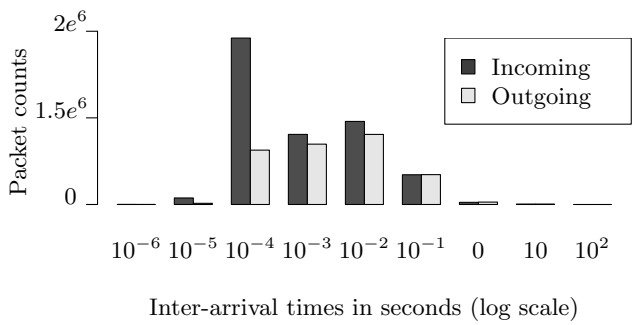


Figure 3: Inter-arrival time histogram in a large sample of the top 35K Alexa.

In order to implement WTF-PAD without revealing distinguishing features between real and fake messages, we need to send dummies in time intervals that follow the same distribution as real messages. In Figure 3, we observe that times for incoming and outgoing traffic have different distributions. The asymmetric bit rates in the connection we used to conduct the crawl account

for this difference. Since WTF-PAD has different histograms in the client and the bridge we can simulate traffic that follows different distributions depending on the direction.

Next, we explain how to find the bursts and the gaps in the inter-arrival time distribution and build the histograms H_B and H_G. Intuitively, the burst-mode histogram H_B should consist of larger delays covering the duration of typical bursts, while the gap-mode histogram H_G should consist of smaller delays that can be used to mimic a burst. To split inter-arrival times into the two histograms, we calculate the instantaneous bandwidth at the time of each inter-arrival time to determine if it is part of a burst or not. Then, we set a threshold on the bandwidth to draw the line between bursts and gaps.

We estimate the instantaneous bandwidth using a sliding window over a sequence of consecutive packets. We have experimented with different window lengths and threshold values. The best results against the state-of-the-art WF attack are achieved for a window of two consecutive packets and a threshold set to the total average bandwidth for the whole sample of traces.

3.4 Tuning mechanism

AP can hide inter-arrival times that are longer than the average, but it does not hide times that are shorter than the average. To effectively hide these times we need to either add delays to exceptionally long traces or add more padding over all traces to level them off and make them less distinctive. We focus on the latter approach because our objective is to minimize delay. WTF-PAD provides a mechanism to tune the trade-off between bandwidth overhead and security: one can modify the parameters of the distributions used to build the histograms to add more padding and react to shorter inter-arrival times.

To illustrate this, we show in Figure 4 the H_B histogram as sampled from our dataset. We observe that the distribution of the logarithm of these times can be approximated with a normal distribution $\mathcal{N}(\mu, \sigma^2)$. That is, the inter-arrival times follow a log-normal distribution. We can modify its mean and variance to obtain another normal distribution $\mathcal{N}(\mu', \sigma'^2)$ that we will use to sample the inter-arrival times of H_B. By using $\mathcal{N}(\mu', \sigma'^2)$ we are shifting the average distribution of inter-arrival times toward shorter values. This results in a greater amount of short times being covered by padding, which increases the bandwidth overhead but causes the pages become less distinguishable and thereby reduces the attacker's accuracy.

We created a statistical model of the underlying distributions of inter-arrival times from the samples we extracted from our dataset. We experimented with

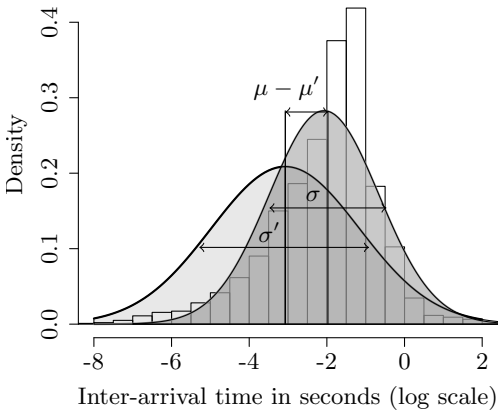


Figure 4: Histogram of times between consecutive bursts for incoming traffic. In dark gray we superpose the PDF of our log-normal fit. In light gray, we show the PDF of a shifted log-normal distribution that we use to build the H_B histogram.

multiple positively skewed distributions to build the model and test the goodness of fit with the Kolmogorov-Smirnov test. We estimated the parameters of the distributions using maximum likelihood estimation. Even though Pareto and Beta distributions seemed to fit best, we decided for simplicity to use normal and log-normal distributions, given that the error was not significantly greater than that observed in the other distributions.

To calibrate the possible shifts, we set μ' and σ' according to the percentile of the real data we want to include. For instance, assuming a normal distribution, if we adjust μ' to the 50th percentile, we obtain $\mu' = \mu$ and $\sigma' = \sigma$. If we set μ' to the value of the Probability Density Function (PDF) at the 10th percentile, we then derive the σ' using the formula of the PDF of the normal distribution.

4 Evaluation

In this section we discuss how we evaluated WTF-PAD, present our findings and compare them with the results we obtained for existing defenses.

4.1 Data

Unlike most previous defense evaluations, which used simulated data, we have used web traffic that has been collected over Tor. We used a dataset that had been collected for a study about a realistic evaluation of WF attacks [10]. This dataset consists of 40 instances, collected in ten batches of four visits, for each *homepage* in top-100 Alexa sites [1]. For the open-world, the dataset also has one instance for each website in the Alexa 35,000 most popular websites.

4.2 Methodology

To evaluate the improvements in performance offered by the defense, we applied the attack's classifier on both the original traffic traces and traces that have been protected by applying the defense. The difference in bandwidth and latency between the original and protected traces provides us with an estimate of the overheads. We applied the state-of-the-art attack on the set of protected traces to evaluate the effectiveness of the defense. The average accuracy over multiple runs determines the security provided by the defense.

In the closed world, we measure the accuracy as the True Positive Rate (TPR), or *Recall*. We also measure the False Positive Rate (FPR), the Positive Predictive Value (PPV)—also called *Precision*, and the harmonic mean of precision and recall (*F1-Score*), as they play an important role on evaluating the effectiveness of the attack in the open-world.

The state-of-the-art attack is based on a k-NN model [22]. k-NN is a supervised learning algorithm that selects the k closest instances (the *neighbors*), and outputs the class of the majority of the neighbors. Wang et al. determined that the number of neighbors that optimizes the trade-off between TPR and FPR is $k = 5$. The distance defined by Wang et al. for use in k-NN is a weighted sum of a set of features. This feature set is the most extensive in the WF literature with more than 4,000 features and including features that extensively exploit bursts.

In order to have a comprehensive evaluation of WTF-PAD, we also evaluated it with other existing WF attacks that take into account features that are not included in k-NN.

4.3 Results

To evaluate the trade-off between bandwidth overhead and accuracy provided by WTF-PAD, we applied the attack on protected traces with different percentile values, ranging from 0.5 (low protection) to 0.01 (high protection) percentiles.

In Figure 5, we show the trade-off curves for both normal and log-normal fits. We observe a steeper decrease in accuracy for the normal model with respect to the log-normal one. Remarkably, beyond a certain point (around 0.1 percentile), the tuning mechanism saturates to 15% accuracy for both models: percentiles lower than that point do not further reduce accuracy and only increase bandwidth overhead. The trend we observe is the cost in bandwidth exponentially growing with the protection level that the defense attempts to provide

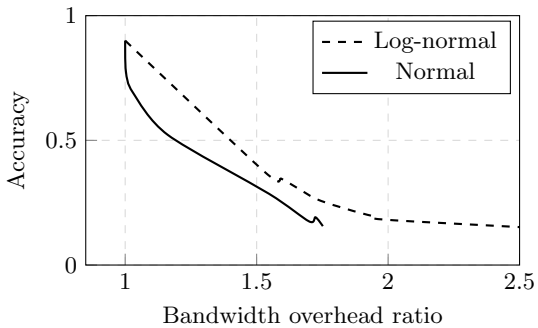


Figure 5: Average accuracy versus median bandwidth overhead ratio.

Table 1 summarizes the security versus overhead trade-off obtained for different attacks (i.e., k-NN, NB, SVM, DL) and defenses BuFLO, Tamaraw, CS-BuFLO and WTF-PAD. As we see, WTF-PAD is the only defense to provide zero latency overhead. The other defenses we tested produce between 145-200% additional average delay to fetch a webpage. WTF-PAD also offers moderate bandwidth overhead. For our datasets, we observed that the bandwidth overhead was always below 60% while attaining decreases in the accuracy of the attack that are comparable with the other defenses.

ROC curve. To study the impact of WTF-PAD on the performance of k-NN, we also plotted the ROC curve with and without protection. The ROC curve represents the performance of the classifier when its discrimination parameter changes. The standard k-NN is not a parametric algorithm, meaning that there is no explicit parameter that one can use to set the threshold and tune the trade-off. We have defined more or less restrictive classifications of k-NN by

Table 1: Performance and security comparison among link-padding defenses in a closed world.

		Accuracy (%)				Overhead (%)	
Defense	Parameters	kNN	Pa-SVM [16]	DL-SVM [23]	VNG++ [7]	Latency	Bandwidth
BuFLO [7]	$\tau = 10$s, $\rho = 20$ms, $d = 1500$B	14.9	14.1	18.75	N/A	145	348
CS-BuFLO [2]	$\rho = [20, 200]$ms, $d = 1500$B, CPSP	N/A	30.6	40.5	22.5	173	130
Tamaraw [22]	$\rho_{out} = 0.053$, $\rho_{in} = 0.138, d = 1500$B	13.6	10.59	18.60	12.1	200	38
WTF-PAD	Normal fit, $p = 0.4$, $d = 1500$B	17.25	15.33	23	26	0	77

setting a minimum number of votes required to classify a page. We used 10-fold cross-validation to average the ROC curve for $k = 5$ neighbors in a closed world of 100 pages. To plot the ROC graph we had to *binarize* the classification: we divided the set of pages into two halves, 50 monitored and 50 non-monitored, and considered the monitored as the positive class and the non-monitored as the negative one. Then, all the positive (monitored) observations that are classified as a page in the positive class are counted as true positives, even if the instances were classified as a *different* monitored page. This is a more advantageous scenario for a surveillance-type of attacker that only tries to identify whether the page is monitored or not.

In Figure 6, we compare the ROC curves for the data before and after applying the defense with respect to random guessing. We notice a significant reduction in the performance of the classifier. Compared to unprotected data with an AUC of 0.95 (close to perfect classification), WTF-PAD has an AUC of 0.66, which is substantially closer to random guessing.

5 Realistic Scenarios

In this section, we present the results of the evaluation of the defense in two realistic scenarios: the open world and the use of multi-tab browsing.

5.1 Open-world evaluation

We now evaluate the performance of the defense against the k-NN algorithm in the open-world scenario. Our definition of the open-world is similar to the ones described in prior work. We have evaluated the k-NN with the evaluation method used by Wang et al. and incorporating the changes suggested by Wang [21], so that we can compare our results with the ones they obtained [22]

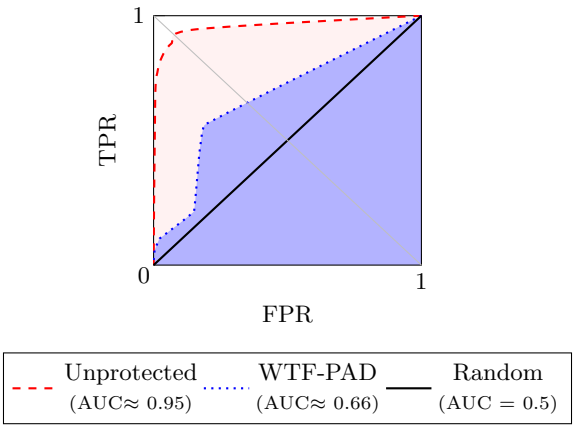


Figure 6: 10-fold cross-validated ROC curves of k-NN with five neighbors and using a strict consensus threshold.

In Wang's open-world classification, they consider one class for each of the monitored pages and one single class for all the non-monitored pages. Then, the attacker aims to identify the exact monitored pages the user visits and to classify all the visits to non-monitored pages into the non-monitored class regardless of the actual page.

We observe that even though the accuracy initially increases as the world grows and saturates to 95% at the maximum considered world size, the F1-Score decreases and levels off to 50%. This is because even though the FPR rapidly drops to zero, the TPR decreases below 40%. The accuracy is so high because the classifier reaches almost perfect classification for the non-monitored class. This high accuracy is due to the stringent threshold used in the k-NN which requires all neighbors to vote to the same class and reduces the FPR.

We observe that the TPR and FPR after applying the defense are dramatically lower than the rates shown in Figure 7. However, due to the skew between the positive and the negative classes, the ROC curves of the k-NN are biased towards the negative class and do not reflect well the performance of the classifier. For imbalanced datasets, it is recommended to use the Precision-Recall ROC (P-ROC) instead of the ROC [5]. Similarly to the standard ROC, P-ROC represents the interaction of TPR (recall) and PPV (precision), instead of FPR, with respect to variations on the discriminant of the classifier. Precision in the open-world scenario conveys the fraction of monitored pages that were correctly detected by the k-NN. Precision is invariant to the size of the negative class and thus gives a more accurate estimation of the classifier's performance in the open-world.

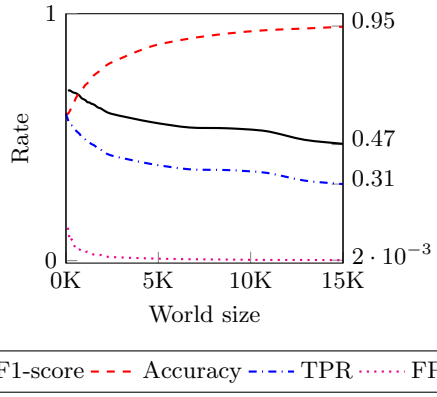


Figure 7: Performance metrics for the classification of a k-NN classifier with $k = 4$ neighbors for an open world up to 15K pages [21].

In the P-ROC graph, the perfect classifier has a curve that coincides with the top-right corner and the random classifier is calculated as the number of positives divided by the total number of instances, i.e., the probability of selecting a positive instance uniformly at random. This random curve is used as a baseline because no classifier can have lower precision than it. As in the standard ROC, classifiers can be bench-marked by comparing their area under the curve (AUC).

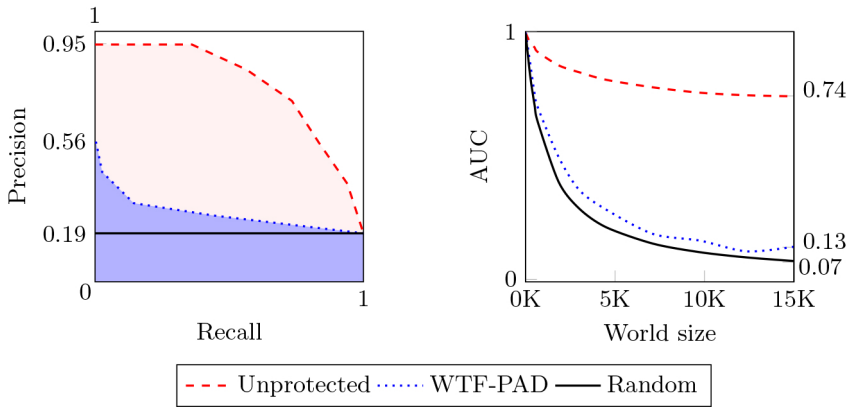


Figure 8: The figure on the left shows the P-ROC curves for the k-NN attack on the protected and unprotected datasets for 5,000 pages. On the right, a comparison of P-ROC AUC with respect to the world size.

Figure 8 (left) shows the P-ROC curve of the k-NN when applied on the set of traces before and after WTF-PAD. Again, we observe that the AUC for the unprotected case is reduced significantly (from 0.79 to 0.27) and is close to random. However, this graph is a snapshot of the performance of the classifier for a fixed world size (5,000 pages). In order to evaluate how the size of the world affects the attack for the unprotected and protected data, we plot in Figure 8 (right) the AUC estimates while varying the size of the world. The first data point represents a closed world where all pages are monitored and, as expected, all classifiers perform as in perfect classification (AUC=1). However, as we increase the size of the world, the baseline classification tends to zero because a random guess is less likely to succeed. The k-NN levels off to AUC 0.74, which means that it is not heavily affected by the size of the world. Notably, when we apply the defense on the traces, all AUC values are close to random even for the largest world size that we have considered (15K pages). WTF-PAD steadily decreases the attack's success at the same rate as the random classifier does.

5.2 Multi-tab evaluation

The objective of the experiments in this section is to evaluate the efficacy of the WTF-PAD defense when the user is browsing with multiple tabs open. For this evaluation, we considered two scenarios and in both, the goal of the attacker is to identify one of the pages that compose the traffic trace.

Table 2: TPR for protected and unprotected traces in Scenarios 1 and 2.

	TPR	
	Unprotected	WTF-PAD
Scenario 1	14%	8%
Scenario 2	68%	22%

In Scenario 1, we trained the k-NN attack on a single-tab dataset and tested on a mixed dataset of single tab traces and multi-tab traces generated by a crawl with two simultaneous tabs. The first tab was loaded following the Alexa top 100 sequentially. The second tab was open with a delay from 0.5 to 5 seconds and was chosen uniformly at random from the same list. Table 2 shows the result of Scenario 1 for traces with and without the protection offered by WTF-PAD.

Since the accuracy of the k-NN is already low when training on single-tab and testing on multi-tab (Scenario 1 in Table 2), the defense does not impact significantly the TPR of the classifier.

Table 3: TPR with respect to each traffic type. Each cell shows the number of background pages (first tab) detected among truly detected multi-tab traces.

	Scenario 1 (TP/Total)			Scenario 2 (TP/Total)		
	Single	Multi	First	Single	Multi	First
Unprotected	233/300	901/8100	544/901	263/300	482/810	449/482
WTF-PAD	95/300	598/8100	333/598	108/300	137/810	103/137

In Scenario 2, we trained and tested k-NN on a dataset that includes multi-tab and single-tab traces. In this scenario, the attack achieves 68% TPR on unprotected multi-tab traces, much higher than the 14% found in Scenario 1. However, the success rate on protected traces drops to 22%.

In Table 3 we group the detection rates by traffic type (single or multi tab) as used to build the test set. k-NN can successfully classify unprotected single-tab traces with an accuracy of 87%, which is close to the accuracy rate of k-NN in the closed-world setting. The accuracy decreases to just 36% when we protect the traces with WTF-PAD.

6 Discussion and future work

WF attacks fall within the Tor threat model [6], as it only requires one point of observation between the client and the bridge or guard, and the attack potentially de-anonymizes users by linking them with their browsing activity. Even with the challenges of open-world and multi-tab browsing [10], some websites may exhibit especially unique traffic patterns and be prone to high-confidence attacks. Attacks may observe visits to the same site over multiple sessions and gain confidence in a result.

Protecting Tor users from WF attacks, however, must be done while maintaining the usability of Tor and limiting costs to Tor relay operators. Delay is already an issue in Tor, so adding additional delay would harm usability significantly. The BuFLO family of defenses add between 145-200% additional delay to the average website download, i.e. up to *three times as long* to get a webpage, which makes them very unlikely to be adopted in Tor.

The main overhead in WTF-PAD is bandwidth, which was under 80% overhead in all scenarios we tested. We do not know the exact percentage that is acceptable for use in Tor, but we note the following points. First, approximately 40% of Tor traffic is bulk downloads (from 2008, the last data we know of) [13].

To the extent that this holds today, only the remaining 60% of traffic needs to be covered by this defense. Second, the bottleneck in Tor bandwidth today is exit nodes. WF defenses do not need to extend to exit nodes, stopping at the bridge (in our framework) or at the guard or middle node when fully implemented. Thus, the bandwidth overhead only extends to one or two relays in a circuit and crucially not to the most loaded relay, making the overhead cost much less in practice. Third, given our findings for the open-world setting, it may be possible to tune WTF-PAD further to lower the bandwidth and maintain useful security gains in realistic use cases.

The construction of the histograms H_B and H_G is critical for the correct performance of the defense. First, since these distributions depend on the client's connection, we cannot estimate them a priori and ship them with WTF-PAD. A solution is to consider groups of clients with similar connections and have a precomputed configuration for each group. Then, the clients will estimate the properties of their network and only download the configuration that best matches their connection. Future work in developing WTF-PAD could explore the use of genetic algorithms to find the optimal histogram for each specific situation. A genetic algorithm could optimize a fitness function composed by the bandwidth overhead and the accuracy of the WF attack. Under mild assumptions on the distribution, histograms can be represented efficiently to reduce the search space.

7 Conclusion

In this paper, we described the design of WTF-PAD, a probabilistic link-padding defense based on Adaptive Padding. We studied the effectiveness and overheads of WTF-PAD, and compared it to existing link-padding-based defenses, showing that it offers reasonable protection with lower overhead costs. In particular, our results show that WTF-PAD does not introduce any delay in the communication while introducing moderate bandwidth overheads, which makes it especially suitable for low-latency communications such as Tor. Additionally, we have evaluated the effectiveness of WTF-AP in open-world and multi-tab scenarios. The results show that the defense reduces the performance of the classifier to random guessing.

References

[1] Alexa. Alexa Top 500 Global Site. `http://www.alexa.com/topsites`, 2015.

[2] Xiang Cai, Rishab Nithyanand, and Rob Johnson. Glove: A bespoke website fingerprinting defense. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 131–134. ACM, 2014.

[3] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *ACM Conference on Computer and Communications Security (CCS)*, pages 227–238. ACM, 2014.

[4] 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.

[5] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

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

[7] 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 *IEEE Symposium on Security and Privacy (S&P)*, pages 332–346. IEEE, 2012.

[8] 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.

[9] Andrew Hintz. Fingerprinting websites using traffic analysis. In *Privacy Enhancing Technologies Symposium (PETS)*, pages 171–178. Springer, 2003.

[10] 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.

[11] Liming Lu, EC Chang, and MC Chan. Website fingerprinting and identification using ordered feature sequences. In *European Symposium on Research in Computer Security (ESORICS)*, pages 199–214. Springer, 2010.

[12] Xiapu Luo, Peng Zhou, Edmond W. W. Chan, Wenke Lee, Rocky K. C. Chang, and Roberto Perdisci. HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows. In *Network & Distributed System Security Symposium (NDSS)*. IEEE Computer Society, 2011.

[13] Damon McCoy, Kevin Bauer, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Shining light in dark places: Understanding the Tor network. In *Privacy Enhancing Technologies Symposium (PETS)*, July 2008.

[14] 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.

[15] Andriy Panchenko, Fabian Lanze, Andreas Zinnen, Martin Henze, Jan Pennekamp, Klaus Wehrle, and Thomas Engel. Website fingerprinting at internet scale. In *Network & Distributed System Security Symposium (NDSS)*, pages 1–15. IEEE Computer Society, 2016.

[16] 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.

[17] Vitaly Shmatikov and Ming-Hsiu Wang. Timing analysis in low-latency mix networks: Attacks and defenses. *European Symposium on Research in Computer Security (ESORICS)*, 2006.

[18] 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.

[19] The Tor project. Users statistics. `https://metrics.torproject.org/users.html`. (accessed: July 20, 2015).

[20] The Tor project. Pluggable Transports. Tor spec: "`https://gitweb.torproject.org/torspec.git/tree/pt-spec.txt`", 2012. (accessed: December 15, 2015).

[21] Tao Wang. *Website Fingerprinting: Attacks and Defenses*. PhD thesis, University of Waterloo, 2016.

[22] 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.

[23] 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.

[24] CV V Wright, F Monrose, and GM M Masson. On inferring application protocol behaviors in encrypted network traffic. 7(2006):2745–2769, 2006.

A Appendices

A.1 WTF-PAD Histograms

A histogram is defined as a disjoint partition of the support of the inter-arrival time distribution $[0, +\infty)$. Each sub-interval, that we call *bin*, is a half-closed interval $I_i = [a_i, b_i)$ with $0 \leq a_i, b_i \leq +\infty$ for all $i = 1, \ldots, n$, where $n \geq 2$ is the total number of bins in the partition. The bin lengths used in the AP histogram increase exponentially with the bin index, namely, the intermediate bins have the following endpoints:

$$a_i = \frac{M}{2^{n-i}}, b_i = \frac{M}{2^{n-i-1}},$$

for $i = 2, \ldots, n-1$. $M > 0$ is the maximum inter-arrival time considered in practice. The first bin is $I_1 = [0, \frac{M}{2^{n-2}})$ and the last bin is $I_n = [M, +\infty)$.

An exponential scale for the bins provides more resolution for values in a neighborhood of zero, which is convenient to represent distributions with heavy positive skew, such as the distribution of inter-arrival times in network traffic.

When we sample from a bin, AP returns a value sampled uniformly from $[a_i, b_i)$, except for the last bin $[M, +\infty)$, in which case AP returns "∞".

In Figure 9, we show a simplified version of the histograms we used in the WTF-PAD instance at the client. The histograms that we use have 20 bins.

Each bin contains a number of tokens k_i. We denote K the sum of tokens in all the bins except the infinity bin, i.e.:

$$K := \sum_{i=1}^{n-1} k_i.$$

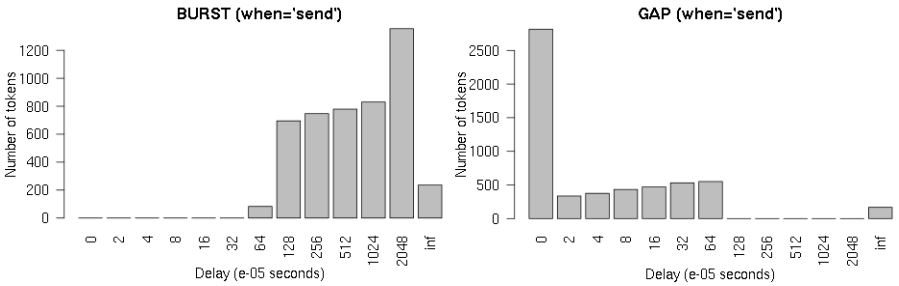


Figure 9: Example of WTF-PAD histograms at the client. The histogram on the top is the H_B and the one at the bottom is H_G.

If we assume the probability of selecting a token is uniform over the total number of tokens, then the probability of sampling a delay from that bin can be estimated as:

$$P_i := \frac{k_i}{K + k_n}. \tag{1}$$

We assume that all the bins I_i for $i < n$ are already filled as explained in Section 3.3. In the following we describe how to set the number of tokens in I_n, the infinity bin, for both histograms, H_B and H_G.

Infinity bin in H_B. According to the notation introduced above, P_n in H_B is the probability of falling into the infinity bin and thus defines the probability of not sending padding (and not starting a fake burst) when we draw a sample from it. To express k_n in terms of the probability of sampling from I_n and the current sum of tokens in the histogram, we clear Equation 1 for k_n:

$$k_n = \frac{P_n}{1 - P_n} K.$$

For instance, if we decide on setting the probability of generating a fake burst to 0.9, then we need to set $P_n = 0.1$. Assuming $K = 300$ tokens, using the equation above we obtain $k_n \approx 34$.

Infinity bin in H_G. The number of tokens we will sample from H_G until we hit the infinity bin is the number of dummy messages we will send within a fake burst. Since the probability of drawing a token is uniform, we can think the histogram as one single bucket that contains tokens from I_n and tokens from the other bins. Then, the expected number of draws without replacement, L,

until we draw the first token from the infinity bin is a known result one can find in any probability textbook:

$$E[L] = \frac{K + k_n + 1}{k_n + 1}.$$

We know the expected value of the length of a burst from our estimations on a large dataset of web traffic. Let μ_L be the mean burst length. In order to make sure fake bursts have the same mean length as real bursts, we must impose the expected number of tokens we sample until we hit the infinity bin to be: $E[L] = \mu_L$. Then, we only need to clear k_n from the equation:

$$k_n = \frac{K - \mu_L + 1}{\mu_L - 1}.$$

This equation is well defined because, typically, the mean length of a burst is small: $K >> \mu_L$.

Publication

Website Fingerprinting Defenses at the Application Layer

Publication Data

Cherubin, G., Hayes, J., and Juarez, M. "Website fingerprinting defenses at the application layer". In *Proceedings on Privacy Enhancing Technologies (PoETS)* (2017), De Gruyter, pp. 168–185

Contributions

- Principal author. All authors contributed equally.

Website Fingerprinting Defenses at the Application Layer

Giovanni Cherubin[1], Jamie Hayes[2], and Marc Juarez[3]

[1] Royal Holloway University of London, Egham, UK
[2] University College London, London, UK
[3] KU Leuven, ESAT/COSIC and imec, Leuven, Belgium

Abstract. *Website Fingerprinting* (WF) allows a passive network adversary to learn the websites that a client visits by analyzing traffic patterns that are unique to each website. It has been recently shown that these attacks are particularly effective against `.onion` sites, anonymous web servers hosted within the Tor network. Given the sensitive nature of the content of these services, the implications of WF on the Tor network are alarming. Prior work has only considered defenses at the client-side arguing that web servers lack of incentives to adopt countermeasures. Furthermore, most of these defenses have been designed to operate on the stream of network packets, making practical deployment difficult. In this paper, we propose two application-level defenses including the first server-side defense against WF, as `.onion` services have incentives to support it. The other defense is a lightweight client-side defense implemented as a browser add-on, improving ease of deployment over previous approaches. In our evaluations, the server-side defense is able to reduce WF accuracy on Tor `.onion` sites from 69.6% to 10% and the client-side defense reduces accuracy from 64% to 31.5%.

1 Introduction

Website Fingerprinting (WF) attacks allow a passive local adversary to infer which webpage a client is viewing by identifying patterns in network traffic that are unique to the webpage. These attacks are possible even if the client is browsing through anonymity networks such as Tor and the communication is encrypted [12]. Tor routes a client's traffic through volunteer relays before connecting to the communication's destination, so that local eavesdroppers cannot link both sender and receiver of the communication [8]. However, the

WF attack, if successful, breaks the *unlinkability* property that Tor aims to provide to its users.

Moreover, a 2015 study has shown that `.onion` sites can be distinguished from regular sites with more than 90% accuracy [16]. This substantially narrows down the classification space in Tor and suggests the attack is potentially more effective at identifying `.onion` sites than regular pages. *Onion services* are websites with the `.onion` domain hosted over Tor, allowing a client to visit a website without requiring it to publicly announce its IP address. These sites tend to host sensitive content and may be more interesting for an adversary, turning the WF attack into a major threat for connecting users. In this paper, we propose the first set of defenses specifically designed and evaluated for Tor `.onion` sites.

WF defenses are often theorized at the network level, and try to disrupt statistical patterns via inserting dummy messages in to the packet stream [3, 4, 9]. Some defenses try to alter the network traffic of a webpage to mimic that of another webpage that is not interesting to the attacker [31]. However, a defense at the network level may require substantial changes of Tor or even the TCP stack, which would make its deployment unrealistic. Furthermore, there is no need to hide patterns at the network layer because few webpage-identifying features, if any, are introduced by low layers of the stack (e.g., TCP, IP). In this work, we consider application-layer defenses, arguing that this approach is more natural for WF defenses and facilitates their development.

Existing WF defenses have been engineered to protect the link between the client and the entry to the Tor network, assuming this is the only part of the network observable by the adversary. We propose both WF defenses at the client- and server-side. A server-side defense is more usable as it does not require any action from the user. More and more, certain types of websites, such as human rights advocacy websites, have the motivation to provide WF defenses as a service to its user base, who may be of particular interest to an adversary. For this reason, we believe that, in contrast to *normal* websites, `.onion` site operators not only have the incentive to provide defenses against WF attacks, but can also achieve a competitive advantage with respect to other `.onion` sites by doing so.

As a real life motivating example, we were contacted by *SecureDrop* [1], an organization that provides onion services for the anonymous communication between journalists and whistleblowers. They are concerned that sources wishing to use their service can be de-anonymized through WF. As a consequence, they are interested in using a server-side WF defense. We have included a SecureDrop website in all the datasets used for the evaluation of defenses.

We introduce two variants of a server-side defense operating at the application layer, which we call *Application Layer Padding Concerns Adversaries* (ALPaCA). We evaluate it via a live implementation on the Tor network. We first crawl over a significant fraction of the total Tor `.onion` site space, retrieving not only the network level traffic information – as is standard in WF research – but also the `index.html` page and HTTP requests and responses. We then analyze the size distribution for each content type, e.g. PNG, HTML, CSS. Using this information, ALPaCA alters the `index.html` of a page to conform to an "average" `.onion` site page. ALPaCA runs periodically, changing the page fingerprint on every user request.

Due to the expected slow adoption of server-side WF defenses, client-side defenses must still be used. We therefore implement a simple client-side WF defense, dubbed *Lightweight application-Layer Masquerading Add-on* (LLaMA), that works at the application layer by adding extra delays to the HTTP requests. These delays alter the order of the requests in a similar way to randomized pipelining (RP) [23], a WF countermeasure implemented in the Tor browser that has been shown to fail in several evaluations [5, 14, 30]. Besides delaying HTTP requests, our defense sends redundant requests to the server. We show most of the protection provided by this defense stems from the extra requests and not from the randomization of legitimate requests.

Our contributions are, as a result of a real life demand, the first implementation of a server-side WF defense and a simple yet effective lightweight client-side defense. With these two approaches we explore the space of application-layer defenses specifically designed to counter WF in `.onion` sites. In addition, we have collected the largest – to the best of our knowledge – dataset of sizes and types of content hosted by Tor `.onion` sites. We provide an evaluation of the overhead and efficacy of our defenses and compare it to some of the most practicable existing WF defenses.

The source code and datasets of both ALPaCA and LLaMA have been made publicly available on GitHub[4]. The original code is also available on an `.onion` site[5], which is protected using our defense.

2 Threat model

As depicted in Figure 1, we consider an adversary who has access to the communication between the client and the entry point to the Tor network, known as *entry guard*. A wide range of actors could have access to such communications,

[4] `http://github.com/camelids/`
[5] `http://3tmaadslguc72xc2.onion`

ranging from malicious or corrupted relay operators, who can target all clients connecting to the guards they control; to ASes, ISPs and local network administrators, who can eavesdrop on Tor clients located within their infrastructure.

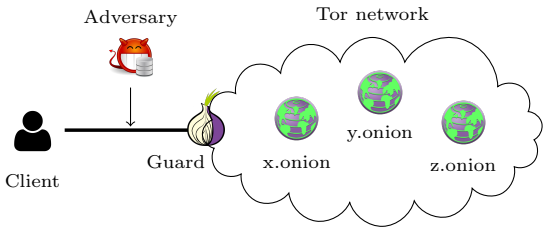


Figure 1: A client visits an `.onion` site over Tor. The attacker eavesdrops the encrypted link between the Tor client and the entry *guard* to the Tor network. Between the client and the destination onion service there is a six-hop Tor circuit that we have omitted to simplify the figure.

The adversary eavesdrops the communication to obtain a *trace* or *sample instance* of the encrypted network packets. He can observe and record these packets, but he cannot decrypt them. Furthermore, we will assume a *passive* adversary: he cannot remove or modify the packets, nor drop or add new packets to the stream.

The objective of the adversary is to infer the websites that were visited by the client from the traffic samples. The adversary can build a template for a number of websites with his own visits and then match the traffic generated by the client. It has been shown that, for a small set of websites, such an attacker can achieve high success rates achieving over 90% accuracy [5].

These attacks have however been criticized for making a number of unrealistic assumptions that favor the adversary [14]. For instance, they assume webpages are static, although some pages have frequent content updates; the client only visits pages that the attacker has trained on, also known as the *closed-world* assumption; and the attacker is able to perfectly parse the fraction of the continuous stream of traffic corresponding to a specific page download, assuming there is a gap between one visit and the next one.

In 2015, Kwon et al. showed that an attacker falling within this threat model can effectively distinguish visits to `.onion` sites from regular websites [16]. They also revisited the assumptions for which prior work on WF had been criticized [14] and found that many of these assumptions hold when considering only `.onion` sites. In contrast to the open Web, the world of `.onion` sites is small and comparable to a closed world, they are also more static than regular websites and their streams are isolated by domain [16]. As in virtually all prior

work on WF, they still assumed the client visits only home pages, ignoring other pages in the website such as inner pages and logged-in or personalized pages that are not available to the attacker. In our evaluation, we follow them and only collect data for the home page of the `.onion` sites we have crawled.

We assume an adversary is only interested in fingerprinting `.onion` sites, and already has a classifier to tell `.onion` traffic apart from the bulk of client traffic. We focus on defenses that protect against the WF attack in the "onion world" because it is a more threatening setting than the one studied in most prior WF work on Tor; visits to `.onion` sites tend to be more sensitive than to pages whose IP address is visible to clients. Luo et al. argue that a WF defense must be implemented at the client-side because web servers have no incentive to offer such a service [19]. However, we believe `.onion` site operators are aware of the privacy concerns that Tor clients have and would make the necessary (minor) modifications in the server to implement our defense.

For the design of ALPaCA, we will assume there is no dynamic content. This includes content generated at the client-side (e.g., AJAX) as well as the server-side (e.g., a PHP script polling a database). This assumption simplifies the design of the server-side defense: ALPaCA requires the size of the web resources being loaded and it is hard to estimate the size of dynamic content a priori.

To assume that no JavaScript will run in the browser is not as unrealistic as it may seem given the high prevalence of JavaScript in the modern Web. The Tor Browser's security slider allows users to select different levels of security, disabling partially or totally JavaScript. Furthermore, SecureDrop pages already ask their clients to disable JavaScript to prevent attacks such as cross-site scripting. It is reasonable to think that clients who protect themselves against WF will first disable JavaScript to prevent these other attacks.

3 Related work

WF is typically modeled as a supervised learning problem. The attacker collects traffic traces for a large sample of websites that aims to identify and builds a classifier that outputs a label, with a certain level of confidence. Since the first WF classifiers were proposed in the late nineties [7], the attacks have been developed with improved classification models to defeat a wide variety of privacy enhancing technologies such as encrypting web proxies [13,27], SSH tunnels [17], VPNs, and even anonymity systems such as Tor and JAP [12].

3.1 Attacks

The latest attacks against Tor achieve more than 90% accuracy in a *closed-world* of websites, where the attacker is assumed to have samples for all the websites a target user may visit [5, 11, 20, 29, 30]. This assumption is unrealistically advantageous for the attacker [14] and a recent study has shown that the attack does not scale to large open-worlds [20]. However, the `.onion` space is significantly smaller than the Web and may be feasible for an adversary to train on a substantial fraction of all `.onion` websites. Furthermore, the closed-world evaluation provides a lower bound for the efficacy of the defense. For a complete evaluation of the performance of our defenses, in this paper we will provide results for both open and closed-world scenarios.

We have selected the most relevant attacks in the literature to evaluate our defenses:

k-NN [29]: Wang et al. proposed a feature set of more than 3,000 traffic features and defined an adaptive distance that gives more weight to those features that provide more information. To classify a new instance, the attack takes the label of the k Nearest Neighbors (k-NN) and only makes a guess if all the neighbors agree, minimizing the number of false positives.

CUMUL [20]: The features of this attack are based on the cumulative sums of packet sizes. The authors interpolated a fixed number of points from this cumulative sum to build the feature vectors that they use to feed a Support Vector Machine (SVM).

k-FP [11]: Hayes and Danezis used Random Forests (RF) to transform, based on the leafs of the trees, an initial feature set to another feature set that encodes the similarity of an instance with respect to its neighbors. Then, they also used a k-NN for final classification.

3.2 Defenses

Most WF defenses in the literature are based on *link-padding*. The traffic morphing approach attempts to transform the traffic of a page to resemble that of another page [18, 21, 31], or to generalize groups of traffic traces in to anonymity sets [4, 29]. The main downside of this type of defenses is that they require a large database of traffic traces that would be costly to maintain [14].

Link-padding aims to conceal patterns in web traffic by adding varying amounts of dummy messages and delays in flows. Link-padding has been used for traffic

morphing to cause confusion in the classifier by disguising the target page fingerprint as that of another page [31]. However, as Dyer et al. note [9], traffic morphing techniques produce high bandwidth overheads as some packets must be buffered for a long period. The strategy we follow in ALPaCA is different from traffic morphing in that page contents are not disguised as other pages', but rather the content is modified to become less *fingerprintable*. The intuition behind ALPaCA is to make each resource look like an "average" resource, according to the distribution of resources in the world of pages. This approach reduces the overheads with respect to morphing, as resizing an object to an average size will tend to require less amount of padding than to an object of a specific page. We have experimented with morphing the contents in a page to make it look like another page. This can be seen as the application-level counterpart of *traffic morphing* and the results can be found in subsection A.1.

In 2012, Dyer et al. presented *BuFLO* [9], a defense based on constant-rate link-padding. Although *BuFLO* is a proof-of-concept defense and has high bandwidth overheads, other defenses have been developed from the original *BuFLO* design. *Tamaraw* [3] and *CS-BuFLO* [4] optimize *BuFLO*'s bandwidth and latency overheads to make its deployment feasible. Both of these defenses address the issue of padding the page's tail. BuFLO padded all pages up to a certain maximum number of bytes producing the high bandwidth overheads. CS-BuFLO and Tamaraw proposed a strategy to pad pages to multiples of a certain parameter, which groups pages in anonymity sets by size and significantly reduces the bandwidth overhead over BuFLO. We follow a similar strategy for one of the modes of ALPaCA.

Recently, a lightweight defense based on Adaptive Padding has also been proposed to counter WF [15]. In order to offer low latency overheads, this defense only pads time gaps in traffic that are statistically unlikely to happen. To empirically determine the likelihood of a gap they sampled a large number of pages over Tor and built a distribution of inter-arrival times used to sample the delays for the dummy messages.

Our main concern with these designs is that padding is applied at the network layer. There is no need to apply the defense at the network layer because layers below HTTP do not carry identifying information about the webpage. One could argue that latency and bandwidth identify the web server. However, these features vary depending on network conditions and are shared by all pages hosted in the same server or behind the same CDN. Moreover, the implementation of such defenses may require modifications in the Tor protocol and even the TCP stack, as they generate Tor cells that are sent over Tor's TLS connections.

Application layer defenses act directly on the objects that are fingerprinted at the network layer. The padding is also added directly to these objects. As

opposed to network-layer defenses that must model legitimate traffic to generate padding, application-layer defenses inject the padding inside the encrypted payload and is, consequently, already indistinguishable from legitimate traffic at the network layer. In addition, defenses at the application layer do not require modifications in the source code of the Tor protocol, which make them more suitable for deployment.

In this paper we present and explore two novel approaches for application layer defenses at both client and server-side. In the rest of this section we describe the state of the art on application-layer defenses.

Server-side

To the best of our knowledge, there is only a prototype of a server-side defense that was drafted by Chen et al. and it was designed for a slightly different although related problem [6]. They studied WF in the context of SSL web applications, where the attacker is not trying to fingerprint websites, but specific pages within one single website. Their main contribution was to show that a local passive adversary can identify fine-grained user interactions within the website. The authors devise a defense that requires modifications at both client and server sides, which allows padding to be added to individual HTTP requests and responses.

Client-side

There are only two application-layer defenses proposed in the WF literature: *HTTPOS* [19] and *Randomized Pipelining* (RP) [23]. Luo et al. proposed *HTTPOS* as a client-side defense arguing that server-side or hybrid defenses would see little adoption in the wild due to lack of incentives [19]. In that study, the authors pinpoint a number of high-level techniques that alter the traffic features exploited by WF attacks. For instance, they modify the HTTP headers and inject fake HTTP requests to modify the length of web object sizes.

RP is the only WF countermeasure that is currently implemented in the Tor browser. It operates by batching together a single clients requests in the HTTP pipeline to randomize their order before being sent to the server. Several studies have applied WF attacks on data collected with a RP-enabled Tor Browser and all of them have shown that the defense was not effective at decreasing the accuracy of the WF attack in the closed world [5, 14, 30]. The reason why RP does not work is not clear and has not been investigated in these evaluations.

4 Defenses

WF attacks are possible because different webpages serve different content. High level features such as the number of requests the browser makes to download a page, the order of these requests and the size of each response, induce distinctive low level features observed in the network traffic [9, 21]. For instance, the number of requests sent by the browser corresponds to the number of objects embedded in the page such as images, scripts, stylesheets, and so on.

Most existing defenses propose to add spurious network packets to the stream to hide these low-level features [3,4,9], However, effectively concealing these features at network level poses technical challenges, as the operation of underlying protocols, i.e. TLS, TCP, IP, obfuscates the relation between low and high level features. For this reason, we believe adding the padding to the actual contents of the page is a more natural strategy to hide traffic features than sending dummy packets: if the defense successfully conceals high-level features, the low-level features will follow.

In this section, we describe in detail the strategies that we propose at the application layer at both server (ALPaCA) and client side (LLaMA) to mitigate WF attacks.

4.1 ALPaCA

ALPaCA is a server-side defense that pads the contents of a webpage and creates new content with the objective of concealing distinctive features at the network level. We demonstrate that this strategy is not only effective, but also practical to deploy. We have implemented and evaluated ALPaCA as a script that periodically runs on a server hosting an `.onion` site.

We first show that it is possible to pad the most popular types of webpage objects (e.g., images, HTML) to a desired size, without altering how they look to a user. We then propose two variants of server-side defenses, referred to as P-ALPaCA and D-ALPaCA. At a high level, the defenses choose, for a page to morph, a suitable list of sizes T, that we call *target*. A target specifies the number and size of the objects of the morphed page; P-ALPaCA and D-ALPaCA differ in how they select such a target. Then, the objects of the original page are padded to match the sizes defined in T. If T contains more elements than the page's objects, then new objects ("padding objects") are created and referenced from the morphed HTML page (Algorithm 2). Figure 2 gives a high level overview of this process.

Padding an object to a target size

This section describes how we can pad most types of objects. It is important to note that an adversary looking at encrypted packets cannot: i) distinguish the type of objects that are being downloaded, ii) infer how much padding was added to such objects or whether they were padded at all. By padding an object directly on the server, we can control how large it will look like at the network level. While this control is not complete (because of compression in the HTTP protocol), experiments show that this discrepancy does not largely affect on our defenses.

Table 1 shows the types of objects that we can pad up to a desired size, and their frequency within the `.onion` site world. To pad text objects (e.g., HTML and CCS) we can add the desired amount of random data into a comment. To pad binary objects (e.g., images), it is normally sufficient to append random data to the end of the file; in fact, the file structure allows programs to recognize the end of the file even after this operation.

We verified that binary files would not be corrupted after appending random bytes to them as follows. We used ImageMagick's `identify` program[6] for verifying the validity of PNG, ICO, JPEG, GIF, and BMP files after morphing. The program only raised a warning "length and filesize do not match" for the BMP file; the image was, nevertheless, unaffected, as it could be opened without any errors. We used `mp3val`[7] to check MP3 files; the program returned a warning "Garbage at the end of the file", but the file was not corrupted, and it could be played. We used `ffmpeg`[8] to verify AVI files; the program did not return any errors or warnings.

It is thus possible to morph the most common object types. We suspect that many other types of object can be morphed analogously, by appending random bytes or by inserting data in comments or unused sections of the type structure. We remark, however, that in experiments we did not remove content we could not morph from webpages.

Morphing a page to a target T

We introduce Algorithm 2, which morphs the contents of a page to match the sizes defined by a target T. The target is selected differently by the two versions of ALPaCA, as presented later, and it defines the size of the objects that the morphed page should have.

[6] `http://www.imagemagick.org/`
[7] `http://mp3val.sourceforge.net/`
[8] `https://ffmpeg.org/`

Table 1: Padding the most frequent objects in `.onion` sites to a desired size. "N.O." stands for "not observed". We assume JavaScript is disabled, although it is possible to morph JS files as shown.

Content type	**Morphing**	**Frequency**
PNG, ICO, JPG, GIF, BMP	Append random bytes to the file.	51%
HTML	Insert random data within a comment "<!–", "–>".	13%
CSS	Insert random data within a comment "/*" "*/".	12%
JS	Insert random data within a comment "/*" "*/".	13%
MP3	Append random bytes to the file.	N.O.
AVI	Append random bytes to the file.	N.O.

The algorithm keeps two lists: M, containing the morphed objects, and P, which keeps track of the sizes in T that have not been used for moprhing an object; both lists M and P are initially empty. The algorithm sequentially considers the objects of the original page from the smallest to the largest; for object o, it seeks the smallest size $t \in T$ which o can be padded (i.e., for which $size(o) \leq t$). Once it has found such a t, it removes all the elements of T smaller than t, and pads o to size t; the elements removed from T at this stage (except t) are put into P. After all the original objects have been morphed, the sizes remaining in T are appended to P. New "padding objects" (objects containing random bytes) are generated according to the sizes in P. We make sure that padding objects will be downloaded by a browser, but will not be shown, by inserting a reference to them in the HTML page as if they were hidden images[9]. Finally, the HTML page itself is padded to a target size by the defense.

P-ALPaCA

P-ALPaCA (Probabilistic-ALPaCA) generates a target by randomly sampling from a distribution that represents real-world `.onion` sites. Specifically, it has access to three probability distributions D_n, D_h and D_s, defined respectively on the number of objects a page has, the size of the HTML page and the size of

[9]To add an invisible object called "rnd.png" to an HTML page we insert `<img src="rnd.png" style="visibility:hidden">'`. The browser will consider this a PNG file and it will download it, but it will not attempt to show it. The file, thus, needs not to respect the PNG format, and it can just contain random bytes.

Algorithm 2 Pad a list of objects to a target

Input: O: list of original objects
T: list of target sizes
Output: M: list of morphed objects

```
M ← [ ]
P ← [ ]
▷ Morph the original objects.
while |M| < |O| do
    o ← arg min o ∈ O}{size(o)}
              {
    ▷ Remove the target sizes smaller than size(o).
    while min(T) < size(o) do
        Remove min(T) from T
        Append min(T) to P
    end while
    if T is empty then
        ▷ Cannot morph O to T
        fail
    end if
    ▷ Note: the current min(T) is larger than size(o)
    t ← min(T)
    m ← o padded to size t
    Append m to M
end while
▷ Add padding objects.
Merge P and T into P
for p in P do
    m ← New padding object of size p
    Append m to M
end for
```

each of its objects. The defense samples a target T using these distributions, and then morphs the original page as shown in Algorithm 2.

We estimated D_n, D_h and D_s using Kernel Density Estimation (KDE) from $5,295$ unique `.onion` websites we crawled. Details about crawling and analysis of these websites are in section 5. In section B we show the resulting distributions D_n, D_h and D_s, and provide details on how we used KDE to estimate them.

The defense first samples the number of objects n for the morphed page according to D_n. Then, it samples the size of the morphed HTML from D_h, and n sizes from D_s which constitute a target T. Finally, it attempts to morph the original

page to T (Algorithm 2); if morphing fails, the procedure is repeated. The algorithm is shown in Algorithm 3.

Because sampling from the distributions can (with low probability) produce very large targets T, we introduced a parameter *max_bandwidth* to P-ALPaCA. Before morphing, the defense checks that the total page size is smaller than or equal to this parameter: $\sum_{t \in T} t \leq max\_bandwidth$. If not, the sampling procedure is repeated.

A simple alternative to sampling from a distribution that represents the present state of the `.onion` world, is to sample the number and size of padding objects uniformly at random. We expect that this alternative approach would also set a maximum bandwidth parameter, which would serve as the upper bound of the size of the morphed page. One could imagine that a naive implementation of this alternative approach which sets a high maximum would cause extremely high bandwidth overheads. However, reducing this maximum parameter would constrain the morphed page to look like a small subsection of the onion world, removing altogether the possibility that the page is morphed to resemble a large `.onion` site. Our approach allows a large maximum bandwidth parameter to bet set while ensuring bandwidth overheads will be low. With our approach, the probability that a small page, say `A.onion`, is morphed to the size of a large `.onion` site, say `B.onion`, directly corresponds to the ratio of the number of `.onion` sites within the `.onion` world that are of an equal size to `B.onion`. Meaning a small `.onion` site can have the entire `.onion` world as an anonymity set while ensuring a low bandwidth overhead.

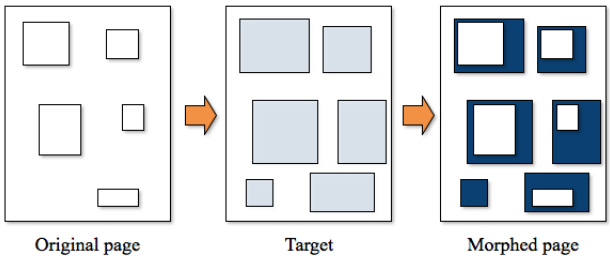


Figure 2: Graphical representation of the server side defenses. Server side defenses P-ALPaCA and D-ALPaCA first select a target for the original web page. Then, they pad the contents of the original page as defined by the target (Algorithm 2), and generate new padding objects if needed. The original and morphed page will look identical to a user.

Algorithm 3 P-ALPaCA

Input: O: list of original objects
D_n: distribution over the number of objects
D_h: distribution over the size of HTML pages
D_s: distribution over the size of objects
html_size: size of the original HTML page
max_bandwidth: maximum page size

▷ We use $x \leftarrow^{\$} D$ to indicate that x is sampled from distribution D
$morphed \leftarrow False$
while not *morphed* **do**
  $T \leftarrow [\,]$
  $h \leftarrow^{\$} D_h$
  if $h <$ *html_size* **then**
    continue
  end if
  $n \leftarrow^{\$} D_n$
  for i in $1..n$ **do**
    $s \leftarrow^{\$} D_s$
    Append s to T
  end for
  if $sum(T) <$ *max_bandwidth* **then**
    Try morphing O to target T (Algorithm 2)
    If successful, $morphed \leftarrow True$
  end if
end while
Pad the HTML page to size h

D-ALPaCA

We propose a second server-side defense, D-ALPaCA (Deterministic-ALPaCA), which decides deterministically by how much a page's objects should be padded. The defense is inspired by Tamaraw [3], which pads the number of packets in a network trace to a multiple of a padding parameter L. D-ALPaCA has the advantage of introducing less overheads than P-ALPaCA, but experimental results suggest this defense is slightly less effective against a WF adversary.

D-ALPaCA (Algorithm 4) accepts as input three parameters: λ, σ and *max_s*, where *max_s* should be a multiple of σ. It pads the number of objects of a page to the next multiple of λ, and the size of each object to the next multiple of σ. Then, if the target number of objects is larger than the original number of objects, it creates padding objects of size sampled uniformly at random

from $\{\sigma, 2\sigma, ..., max\_s\}$. Experiments in section 6 evaluate how different sets of parameters influence security and overheads.

Algorithm 4 D-ALPaCA

Input: O: list of original objects
σ: size parameter
λ: number of objects parameter
$html\_size$: size of the original HTML page
max_s: maximum size of a padding object (should be a multiple of σ)

▷ We use $x \leftarrow^{\$} S$ to indicate that x is sampled uniformly at random from a set S
$T \leftarrow [\,]$
$h \leftarrow$ next multiple of σ greater or equal to $html\_size$
for o in O **do**
    $s \leftarrow$ next multiple of σ greater or equal to $size(o)$
    Append s to T
end for
$n \leftarrow$ next multiple of λ greater or equal to $size(O)$
while $size(T) < n$ **do**
    $s \leftarrow^{\$} \{\sigma, 2\sigma, ..., max\_s\}$
    Append s to T
end while
Morph O to target T (Algorithm 2)
Pad the HTML page to size h

Practicality of the defenses

Both P-ALPaCA and D-ALPaCA are practical to use in real-world applications. In fact, they only require a script to morph the contents of a page periodically. This can be done by setting up a `cron` job running the defense's code, which we release.

Since it is preferable to morph a page after each client's visit, and it may be difficult for the server operator to decide how frequently they should run the `cron` job, we propose a more sophisticated (and flexible) alternative. The defense should preemptively morph the web page many times, and place the morphed pages within distinct directories on the server. Then, the server should be configured to redirect every new request to a different directory. Once the content of a directory has been loaded, the directory is removed, and a new one can be created.

Third-party content

A limitation of ALPaCA is that it can only pad resources hosted in the web server, thus content linked from third parties cannot be protected. In the evaluation of the defense, we have intentionally omitted all third-party content because only two out of the 100 pages in our dataset had resources from third parties.

To understand the impact of this assumption on a larger scale, we have analyzed the prevalence of third-party resources in a crawl of 25K `.onion` sites: only 20% of these sites create requests to third-party domains. Furthermore, for half the pages with third-party content, the third-party requests account for less than 40% of total requests observed within a webpage. However, we found a handful of sites that had more than 90% of their content hosted in third parties. They seem to act as proxies to existing websites. With such a high percentage of unprotected content, the defense is most likely to fail at providing protection against website fingerprinting.

Since the average cost in terms of disk space is 5MB, a possible solution for sites with a large proportion of third-party content would be to cache the third-party resources in the server running the defense. We strongly discourage this approach as if not implemented properly, the `.onion` site, attempting to keep these resources updated, may become vulnerable to timing correlations attacks by the third parties serving the content. In fact, we recommend `.onion` site operators minimize the amount of third-party content they embed to their pages and only cache static content that does not require periodic updates.

4.2 LLaMA

LLaMA is inspired by Randomized Pipelining (RP) [23]. RP modifies the implementation of HTTP pipelining in Firefox to randomize the order of the HTTP requests queued in the pipeline. However, RP has been shown to fail at thwarting WF attacks in several evaluations [5, 14, 30].

LLaMA is implemented as an add-on for the Tor browser that follows a similar strategy to RP: it alters the order in which HTTP requests are sent. The main advantage of a WF defense as a browser add-on is ease of deployment: it does not require modifications to the Tor source code. Thus, a user can install the add-on to enable the protection offered by the defense independently or, if the Tor Project decides to, it could be shipped with the Tor Browser Bundle.

RP exposes a debug flag that logs extra information about its use of the HTTP pipeline [22]. A dataset collected with this flag enabled, visiting the same webpages that the aforementioned evaluations did, provided evidence of a

suboptimal usage of the HTTP pipeline by RP [24]. Either the design of those pages or the low adoption of HTTP pipelining on the servers of these pages or CDNs in between may account for the low performance of RP [2]. Since our defense does not depend on HTTP pipelining, it allows us to test whether these hypotheses hold or it is actually the randomization strategy which is flawed.

Delaying requests. In order to randomize the order of the HTTP requests, the add-on intercepts all requests generated during a visit to a website and adds a different random delay to each one (see Figure 3). We use the statistics extracted from subsection 5.2 to set the distribution of delays for the requests. We take the median page load time in our crawl and set a uniform distribution from zero to half the median load time. As a result, on average, each request will be delayed within a window of half the page load time. In the worst case, this approach will introduce 50% latency overhead if the last request is delayed by the maximum time in the distribution.

Extra requests. As shown in Figure 3, every time a request is sent or a response is received, the extension can be configured to send an extra request. It tosses a coin to decide whether to make an additional HTTP request or not. These fake HTTP requests are sent to a web server that serves custom-sized resources: a parameter in the URL indicates the size of the resource that will be sent in the response body. This allows us to fake random responses from the client-side. Tor isolates streams in different circuits per domain, since such fake requests are made to a different domain they will be sent through a different circuit. This should not be a problem because the attacker cannot distinguish them from legitimate third-party requests. However, as we discuss in the following section, third-party content in `.onion` sites has low prevalence. In addition, this approach requires a trusted server that can be queried from the add-ons. To avoid these issues, the extension implements an alternative method to generate extra responses: it keeps a hash table with domains as keys and lists of request URLs sent to that domain during a browser session as values. To generate a new request, it uniformly samples a URL from the list corresponding to the current first-party domain and sends a request to that URL.

To change the size of legitimate requests we would require cooperation of the server. We acknowledge that previous defenses have proposed this approach [6], but our focus for this defense is to not require any change at the server-side.

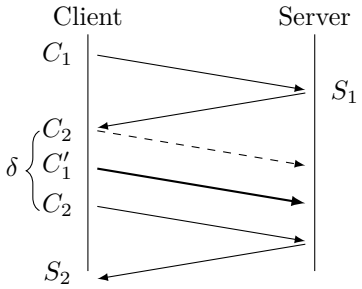


Figure 3: Graphical representation of the LLaMA's operation. δ is the delay added to C_2. C_1', in bold, requests the same resource as C_1.

5 Methodology

In this section we describe the methodology that we followed to collect the data and evaluate the defenses. This data was also used to create the probability distribution used by P-ALPaCA.

5.1 Data collection

For the collection of the dataset we used the `tor-browser-crawler`[10], a web crawler that provides a driver for the Tor Browser, allowing the automation of web page visits in conditions similar to those of regular Tor users. We added support for the Tor Browser Bundle 5.5.5, the latest version at the time of our crawls (March 2016) and extended the crawler to intercept all HTTP requests and responses for future inspection. The crawler logs the size and the URL for each HTTP request and response. The crawler also allows to modify browser preferences. We used this feature to disable JavaScript and RP when needed.

We crawled a list of `.onion` sites obtained from *Ahmia*[11], the most popular search engine for onion services. Ahmia maintains a blacklist of illegal `.onion` sites and thus are excluded from our crawls. The crawl consisted of $25,000$ `.onion` instances, after removing time-outs and failed loads, we captured $18,261$ instances of an `.onion` site load from $5,295$ unique addresses. This dataset serves as both the basis for which we conduct WF attack experiments with our defense in place, as a source of information when inferring the distribution of objects that

[10] `https://github.com/webfp/tor-browser-crawler`
[11] `https://ahmia.fi`

the server-side defense should conform to, and as a source of load time statistics for which the client-side defense decides when to inject additional requests.

5.2 Data analysis

From the 18,261 instances, a total of 177,376 HTTP responses and 7,095 HTTP requests were captured. The average amount of uploaded data per `.onion` site was 256B, while the median amount of uploaded data per `.onion` site was 158B. The average amount of downloaded data per `.onion` site was 608KB, while the median amount of downloaded data per `.onion` site was 45KB. The average size of one response was 55KB; the average size of a request was 87B. Clearly the amount of downloaded data surpasses the amount of uploaded data as clients are simply issuing a HTTP request for objects within the server.

The average number of requests to an `.onion` site was 3, while the average number of responses was 11.

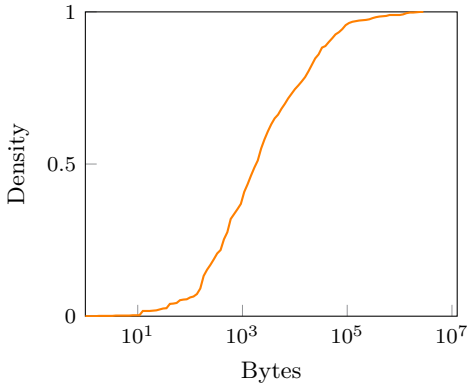


Figure 4: CDF of the HTTP response size in the 25K crawl (in log scale).

The average size of an `.onion` site then is a little over 608KB. In 2015, the average standard website was just over 2MB, and the average number of objects was over 100 [25,28], much larger than the average size and number of objects of an `.onion` site. Clearly there is a distinct difference between standard websites and `.onion` sites; standard websites are much larger and contain a greater number of objects within the HTML index page, we note however that the space of all standard websites is orders of magnitude greater than the space of

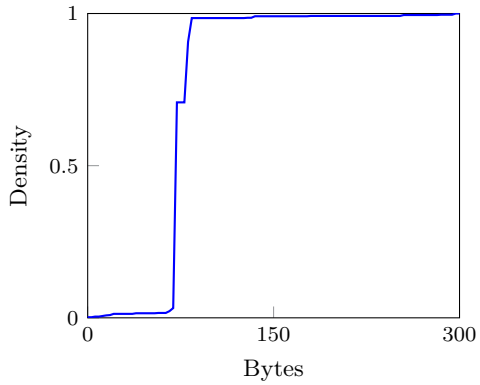


Figure 5: CDF of the HTTP request size in the 25K crawl.

all `.onion` sites and so contains much greater variance in both size and number of objects.

From Figure 5 we see that nearly all HTTP requests were less than 100 bytes, combining this with the knowledge that there are on average just three HTTP requests to download the `.onion` site, we can infer it is most common to download the entire site with just one or two requests after the initial HTTP GET request. From Figure 4, 99% of HTTP responses are less than 1MB in length, and nearly 70% are less than 10KB.

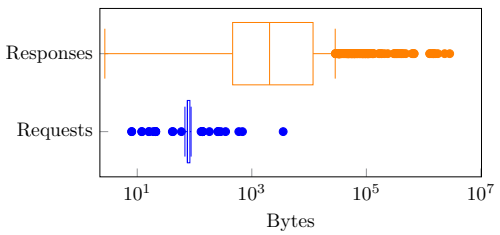


Figure 6: Boxplot of the HTTP request and response sizes for 25K `.onion` sites.

From Figure 6 we see that the majority of requests are between $70 - 100$B, with relatively few outliers. There is a large skew between the majority of responses of size less than a few KB's and a comparatively (to the number of request outliers) large number of response outliers that are orders of magnitude larger in size than the average response size.

6 Evaluation

To assess the effectiveness of our defenses against WF attacks, we have crawled the same set of pages with and without the defenses in place. Comparing the accuracy of state-of-the-art attacks on both datasets provides an estimate of the protection offered by the defenses.

6.1 P-ALPaCA & D-ALPaCA evaluation

We evaluate the server-side defenses when a server does not wish to transform its network traffic to look like another `.onion` site but wishes to morph their traffic so it resembles an "average" `.onion` site. We use results from subsection 5.2 to extract information such as the average number of objects and the average size of these objects across all `.onion` sites. A participating server can then use such information to modify their `index.html` page, resulting in an `.onion` site resembling, at the network layer, many different `.onion` sites rather than a specific targeted site.

The object distributions statistics may change over time and require periodic updates. However, to determine whether they change and how often is out of the scope of this paper and leave it for future research. Such an update mechanism could be served by a trusted entity in the Tor network (e.g., a directory authority) that supplies `.onion` sites with this information.

In addition to transforming the network traffic of an `.onion` site to resemble many different "average" `.onion` sites rather than a targeted site, this method allows the server to control the bandwidth overheads at a more fine grained level, since the server can decide the amount and size of extra objects placed in the `index.html` page.

Table 2: P-ALPaCA & D-ALPaCA latency and bandwidth overheads.

	Latency		Volume	
	%	Avg. (s)	%	Avg. (KB)
Undefended	–	3.99	–	175
P-ALPaCA	52.6	6.09	86.2	326
D-ALPaCA (2, 500, 5000)	66.3	6.63	3.66	182
D-ALPaCA (2, 5000, 5000)	56.1	6.22	9.84	193
D-ALPaCA (5, 2500, 5000)	61.7	6.44	15.1	202
D-ALPaCA (10, 5000, 5000)	41.7	5.65	44	254

The server also has control over how often their site is morphed. The frequency of morphing depends on the estimation of how quickly an adversary can mount an attack. If an adversary, can train on network traffic from the server and monitor during a period where the site remains unchanged, the defense will not be of any use. However, the time to train and launch an attack on a number of `.onion` sites will likely be in the order of hours not minutes[12], as long as a server morphs the site in a shorter period than this, the training data the attacker gathers will be of little use.

To confirm this assertion, we collected 40 network traffic loads, which we call an instance, for each site of 100 `.onion` sites. We chose 100 `.onion` sites that resembled the average size of an `.onion` site[13], in terms of total page size and number of objects. We also collected 40 P-ALPaCA morphed instances for each of the `.onion` sites, such that each instance is the result of a new morphing process[14] . We then check whether an adversary, training on different morphed versions of an `.onion` site, can still correctly determine the `.onion` site of origin.

More specifically, for each of the 100 `.onion` sites, we collect 40 instances. Resulting in 4000 overall traces. We then apply our server-side defense and re-visit the newly defended sites, resulting in another 4000 traces. We then apply, separately, WF attacks to both undefended and defended `.onion` sites, training on 60% of traces and testing on the remaining 40%. We consider the defense successful if the WF attack accuracy on the defended `.onion` sites is dramatically lower than attack accuracy on the undefended `.onion` sites.

To explore the parameter space, we also evaluated D-ALPaCA, under four different parameter choices. We collected 20 instances for the same 100 `.onion` sites and compared attack accuracy against both the undefended and P-ALPaCA defended `.onion` sites. The parameter choices were: λ - the defended page will have a multiple of λ objects, σ - each of the defended page's objects will have a size which is multiple of σ, max_s - when generating new padding objects, sample uniformly within the set [σ, 2*σ, 3*σ, ..., max_s]. Specifically, we chose the following parameter values for (λ, σ, max_s): $(2, 500, 5000)$, $(2, 5000, 5000)$, $(5, 2500, 5000)$, $(10, 5000, 5000)$.

User Experience: in Table 2, we see that average latencies are approximately 40-60% greater in the protected traces than in the unprotected ones. In seconds,

[12]For example, we used a total of 100 `.onion` sites in experiments, visiting each `.onion` sites 40 times. We trained on 60% of data. The average page load time was around 4 seconds. Therefore an attacker, using one machine for crawling and gathering training data, would be able to initiate an attack after 9600 seconds. However, we note an attacker can parallelize this process for faster attacks.

[13]Via section 5.

[14]As proposed in subsection 4.1, the `.onion` site is morphed on every client visit.

the extra time that the user will spend loading the pages is between two and three seconds. We also measured the times to load the original resources in the protected traces with respect to loading all content, since serving extra padding resources once all the original content is sent does not impact on user experience. We call the time between the first request to the last legitimate request *UX-time*. However, the average difference between UX-time and the time to load all resources in a protected page is less than 200ms. We notice that the randomization of RP often sends original requests at the end of the transmission which explains the mild difference between UX-time and total page load time.

Table 3: Closed world classification for `.onion` sites morphed via P-ALPaCA and D-ALPaCA, with other defenses added for comparison. CUMUL depends on packet lengths and so some defenses that only operate on packet time information cannot be applied.

	k-NN (%)	k-FP (%)	CUMUL (%)
Undefended	45.6	69.6	55.6
P-ALPaCA	0.2	9.5	15.6
D-ALPaCA (2, 500, 5000)	9.5	22.7	27.0
D-ALPaCA (2, 5000, 5000)	12.5	34.4	40.0
D-ALPaCA (5, 2500, 5000)	5.8	22.3	30
D-ALPaCA (10, 5000, 5000)	7.2	22.9	33.0
Decoy [21]	4.9	11.2	X
Tamaraw [3]	6.8	14.0	X
BuFLO [9]	5.3	13.3	X

Closed World classification: we performed a closed world WF attack on P-ALPaCA defended, D-ALPaCA defended and undefended `.onion` sites. If our server-side defenses are successful, defended `.onion` sites should, at the network level, look similar to one another and result in a low classification accuracy. We use CUMUL [20], k-FP [11] k-NN [29] for evaluation[15]. The number of neighbours used for classification is fixed at two.

Table 3 shows the closed-world classification results of undefended `.onion` sites against `.onion` sites with each instance uniquely defended using P-ALPaCA or D-ALPaCA. WF attacks are ineffective under both defenses, and in fact P-

[15]We use Tobias Pulls' implementation of the k-NN website fingerprinting attack [26].

ALPaCA improves upon *Tamaraw* and *BuFLO*. D-ALPaCA does slightly worse than the P-ALPaCA in terms of defending `.onion` sites, but as can be seen from Table 2, has real advantages in terms of limiting bandwidth overheads. For example, D-ALPaCA with parameters (2, 500, 5000), reduced k-FP accuracy from 69.6% to 22.7%, compared to the P-ALPaCA which reduced attack accuracy to 10%. But, D-ALPaCA (2, 500, 5000) required 23.6 times less bandwidth than P-ALPaCA to achieve these results. A server operator wishing to provide a defense to its clients while limiting the increase in bandwidth may then consider this a worthwhile trade-off and choose to use D-ALPaCA over P-ALPaCA.

Table 4: Open world classification for `.onion` sites morphed with P-ALPaCA and D-ALPaCA.

	k-NN (%)		k-FP (%)		CUMUL-k-FP (%)	
	TPR	FPR	TPR	FPR	TPR	FPR
Undefended	37.0	1.0	62.1	0.8	49.7	5.4
P-ALPaCA	0.4	0.2	3.6	0.2	1.1	1.3
D-ALPaCA (2, 500, 5000)	4.5	0.2	12.0	0.4	21.4	1.4
D-ALPaCA (2, 5000, 5000)	7.5	0.4	12.6	0.4	28.8	1.2
D-ALPaCA (5, 2500, 5000)	6.0	0.3	12.7	0.3	18.7	1.3
D-ALPaCA (10, 5000, 5000)	3.4	0.3	13.3	0.3	27.3	1.0

Open World classification: in addition to closed world experiments, we evaluated the server-side defenses in the open world setting, where we include network traffic instances of `.onion` sites that are not of interest to the attacker. We observe how the classification accuracy is affected in this setting, which is intended to reflect a more realistic attack. We use $5,259$ unique `.onion` sites, from subsection 5.2, as background traffic instances[16] and set the number of neighbours used for classification at two. Note that CUMUL only does binary classification in the open world, classifying as either a background instance or a foreground instance of interest, whereas k-FP and k-NN attempt to classify an instance to the correct `.onion` site if it is flagged as a non-background instance. In order to compare the results of the attacks in the open-world, we have used the feature vectors of CUMUL while applying the k-FP classification process. To make sure that the classification model does not affect the accuracy of the attack, we evaluated the CUMUL features with k-FP in a closed-world and achieved a similar accuracy to SVM.

[16] For k-FP, we train on 1,000 of the $5,259$ background traces and, for each `.onion` site, on 50-75% of instances. Whereas k-NN uses Leave-one-out cross-validation on the whole dataset.

As we can see from Table 4 there is a dramatic decrease in attack accuracy when both P-ALPaCA and D-ALPaCA are used, showing that if a server morphs their site at a higher rate than the adversary can gather training data, the site will be almost perfectly concealed.

D-ALPaCA parameter choices: Table 3 and Table 4 show there is no notable difference in attack accuracy when changing parameters. However, as expected, smaller parameter choices led to smaller bandwidth overheads.

6.2 LLaMA evaluation

We have crawled the same list of `.onion` sites as in the evaluation of ALPaCA, under four different conditions:

JS enabled: we collected our data with no defense installed and JavaScript enabled, the default setting in the Tor Browser.

JS disabled: we repeated the same crawl as with JS enabled but disabling JavaScript in the Tor Browser. We keep JS disabled for the rest of our crawls.

RP with delays: we collected data with the defense only delaying requests, altering the order of the requests as described in section 4.

Extra requests: we crawled the same data with the defense adding delays and extra requests as described in the previous section.

We note that we have disabled RP in the Tor Browser for all the crawls above by disabling the browser preference `network.http.pipelining`.

In Table 5, we show the results for the three classifiers in the closed world of 100 onion sites. We do not observe much difference in accuracy between JavaScript enabled and disabled. This shows that our assumption of no dynamic content holds for the list of onion sites used in our evaluation.

When the defense only adds delays to requests, the accuracy of the classifiers decreases 10% in the k-NN classifier and has limited effect on k-FP and CUMUL. The mild impact on the accuracy of the classifier may imply that the hypothesis that RP does not work because servers do not support HTTP pipelining does not hold, suggesting that the request randomization strategy is flawed, as previous evaluations have argued [5, 30].

Table 5: Closed world classification for `.onion` sites under different countermeasures.

	k-NN (%)	k-FP (%)	CUMUL (%)
JS enabled	64.0	55.8	52.4
JS disabled	60.8	53.4	52.7
RP with delays	46.8	47.9	49.6
Extra requests	31.5	36.0	34.8

We also evaluated the scenario in which the countermeasure, besides adding delays, repeats previous HTTP requests. We observe a significant decrease in accuracy to almost half the accuracy obtained in the unprotected case for the k-NN classifier.

In Table 6, we show the overheads of LLaMA for its two different modes. We see that overheads are around 10%. Even though the protection provided by the defense is considerably lower than the server-side defense or other defenses in the literature, its simplicity and the small overhead that it introduces makes it a good candidate for a WF countermeasure.

Table 6: Latency and bandwidth overheads of the client-side defense in the closed world.

	Latency		Volume	
	%	Avg. (s)	%	Avg. (KB)
JS disabled	–	5.01	–	126
RP with delays	8.4	5.42	X	X
Extra requests	9.8	5.49	7.14	135

7 Discussion and future work

Both the ALPaCA and LLaMA have performed at least as well as state-of-the-art defenses, showing that application layer WF defenses do indeed protect against attacks. Next we discuss potential avenues for future research.

Ease of Deployment. We argue that application layer defenses are simpler to implement than previously proposed approaches as they require no modifications to existing protocols or participation from a relay in the circuit. The only expensive part of ALPaCA comes in the form of the gathering of statistics for the probabilistic based morphing approach. However, we suggest this cost can be amortized over all participating servers by allowing a centralized entity to collect this information, such as is done by directory authorities now to collect Tor relay descriptors. Future research could determine how often these statistics must be updated. Implementation of the client-side defense is simple, as we developed it as a browser add-on. This could be made available to Tor clients either by direct integration in to the Tor browser bundle, or through an add-on store.

Rate of Adoption. Initially, we expect relatively few `.onion` sites to implement server-side defenses. Over time if a significant number of `.onion` sites adopt ALPaCA, it is possible that a large fraction of sites will morph their page to resemble one another. In turn, this will create stable anonymity sets of `.onion` sites that have the same network traffic patterns. Finding the rate and size of these anonymity sets is left for future work.

Clearly, smaller `.onion` sites are easier to protect than larger ones, as it is impossible to morph a larger site to resemble network traffic patterns of a smaller site. Thus, we expect larger `.onion` sites to be more difficult to protect over time. However, as subsection 5.2 show, the majority of `.onion` sites are small and so should be relatively simple to defend against WF attacks.

Latency and Bandwidth Overheads. All WF defenses come at the expense of added latency and bandwidth. Our defenses allow the exact overheads to be tuned by the participating client or server. We saw from subsection 6.1 that P-ALPaCA adds, on average, 52.6% extra waiting time and 86.2% additional bandwidth. We note, that compared to previous works, these overheads are relatively small, and that due to the nature of `.onion` sites, even the morphed pages are small in size compared to standard web pages. LLaMA improves on striking a balance between overhead limitation and protection against WF attacks. By issuing additional HTTP requests, WF attack accuracy is halved, while only adding 9.8% in waiting time and 7.14% in bandwidth. We also saw comparably small overheads in our D-ALPaCA defense which significantly reduced WF attack accuracy at the expense of an additional 3.66% of bandwidth.

Natural WF Defenses. We note that compared to related works, the attack accuracy on `.onion` sites seems alarmingly low. Wang et al. [29] achieved accuracies of over 90% when fingerprinting the top 100 Alexa websites, whereas our experiments on 100 `.onion` sites resulted in an accuracy of only 45.6% using the same classifier. We have validated the results of Wang et al. on the top

100 Alexa websites, removing the possibility of a bug or some irregularity in our own crawler. We conclude that this reduction in accuracy is an artifact of the size and type of the majority of `.onion` sites. The average size of a `.onion` site is substantially smaller than that of a standard web page; resulting in less information being leaked to a classification process, allowing for the increase in chance of misclassifications. We also found that a large number of `.onion` sites are log-in pages to various forums, that are based on standard designs and so bear a resemblance to one another. The small size and design of `.onion` sites provide a natural defense against WF. By restricting the amount of information available to a classification process, and conforming to standard website designs, despite the small world size of `.onion` sites we conclude that successful website fingerprinting attacks are considerably more difficult than on standard websites.

HTTP/2. HTTP/2 is the upcoming new version of the HTTP protocol and is already supported by some of the domains that receive most traffic volume in the Web [2]. HTTP/1.1 tried to provide parallelism of HTTP messages with HTTP pipelining. However, the deployment of HTTP pipelining has not been ideal, as many intermediaries (e.g., CDNs) do not implement it correctly [2]. HTTP/2 supports parallelism of HTTP conversations natively and overcomes one of the main limitations of HTTP/1.1. From our experiments with request randomization performed with LLaMA, our intuition is that randomization of HTTP/2 will not provide better results than RP. HTTP/2 also allows to add padding in HTTP messages to mitigate cryptographic attacks [10]. We devise the use of HTTP/2 padding as a primitive for application-layer WF defenses.

8 Conclusion

We proposed two WF defenses for `.onion` sites, a server-side defense and a client-side defense, that operate at the application layer. The choice of working at this layer has the following benefits: i) it gives fine control over the content of webpages, which is arguably the reason why WF attacks are possible, and ii) it makes the defenses easy to implement.

The server-side defenses morph the content of a webpage before it is loaded by a client. The resulting webpage looks exactly as the original in terms of its visual content, but it behaves as a completely different page at the network level, where the WF adversary sits. Intuitively, since the adversary will observe a different webpage for each load, they will not be able to perform the attack. Experiments on `.onion` sites confirm this intuition, and show that this defense effectively reduces the accuracy of an adversary.

We have designed and evaluated a lightweight client-side defense at the application layer. The evaluation shows that this defense reduces the accuracy of the attack in the onion world significantly and, even though it offers lower protection than the server-side defenses, it provides a high security versus overhead ratio. Furthermore, its simplicity and its implementation as an add-on for the Tor Browser favor its deployment in the live Tor network.

References

[1] SecureDrop: the open-source whistleblower submission system. `https://securedrop.org`.

[2] HTTP/2 specs. "`https://http2.github.io/`", 2015. (accessed: August, 2016).

[3] Xiang Cai, Rishab Nithyanand, and Rob Johnson. CS-BuFLO: A congestion sensitive website fingerprinting defense. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 121–130. ACM, 2014.

[4] Xiang Cai, Rishab Nithyanand, and Rob Johnson. Glove: A bespoke website fingerprinting defense. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 131–134. ACM, 2014.

[5] 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.

[6] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *IEEE Symposium on Security and Privacy (S&P)*, pages 191–206. IEEE, 2010.

[7] 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`.

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

[9] 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 *IEEE Symposium on Security and Privacy (S&P)*, pages 332–346. IEEE, 2012.

[10] Yoel Gluck, Neal Harris, and Angelo Prado. Breach: reviving the crime attack. *Unpublished manuscript*, 2013.

[11] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *USENIX Security Symposium*, pages 1–17. USENIX Association, 2016.

[12] 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.

[13] Andrew Hintz. Fingerprinting websites using traffic analysis. In *Privacy Enhancing Technologies Symposium (PETS)*, pages 171–178. Springer, 2003.

[14] 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.

[15] 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 (ESORICS)*, pages 27–46. Springer, 2016.

[16] Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas. Circuit fingerprinting attacks: Passive deanonymization of tor hidden services. In *USENIX Security Symposium*, pages 287–302. USENIX Association, 2015.

[17] 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.

[18] Liming Lu, EC Chang, and MC Chan. Website fingerprinting and identification using ordered feature sequences. In *European Symposium on Research in Computer Security (ESORICS)*, pages 199–214. Springer, 2010.

[19] Xiapu Luo, Peng Zhou, Edmond W. W. Chan, Wenke Lee, Rocky K. C. Chang, and Roberto Perdisci. HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows. In *Network & Distributed System Security Symposium (NDSS)*. IEEE Computer Society, 2011.

[20] Andriy Panchenko, Fabian Lanze, Andreas Zinnen, Martin Henze, Jan Pennekamp, Klaus Wehrle, and Thomas Engel. Website fingerprinting at internet scale. In *Network & Distributed System Security Symposium (NDSS)*, pages 1–15. IEEE Computer Society, 2016.

[21] 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.

[22] Mike Perry. Committed to the official Tor Browser git repository, `https://gitweb.torproject.org/tor-browser.git/commit/?id=354b3b`.

[23] Mike Perry. Experimental defense for website traffic fingerprinting. Tor Project Blog. `https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting`, 2011. (accessed: October 10, 2013).

[24] Mike Perry, Gunes Acar, and Marc Juarez. personal communication.

[25] Alex Pinto. Web Page Sizes: A (Not So) Brief History of Page Size through 2015. yottaa.com. "`http://www.yottaa.com/company/blog/application-optimization/a-brief-history-of-web-page-size/`", 2015. (accessed: April 18, 2016).

[26] Tobias Pulls. A golang implementation of the kNN website fingerprinting attack. "`https://github.com/pylls/go-knn`", 2016. (accessed: May, 2016).

[27] 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.

[28] MobiForge. mobiforge.com. "`https://mobiforge.com/research-analysis/the-web-is-doom`", 2016. (accessed: April 20, 2016).

[29] 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.

[30] 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.

[31] Charles V Wright, Scott E Coull, and Fabian Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *Network & Distributed System Security Symposium (NDSS)*. IEEE Computer Society, 2009.

A Appendices

A.1 Onion service target experiments

In addition to morphing a page via P-ALPaCA and D-ALPaCA, we evaluate the efficacy of our server-side defense on a number of `.onion` sites via morphing to a target `.onion` site. We dispense with applying a new morphing process for each capture of an `.onion` site load. Instead, we morph the `.onion` site once and capture 40 instances of this morphed site. We show that even if a server morphs their network traffic once, if it is morphed towards a targeted `.onion` site, this is enough to thwart WF attacks.

SecureDrop

To protect its users, SecureDrop may want to morph the network traffic pattern of its page load to look like that of an `.onion` site which would not raise suspicion on a visit. We collected 40 instances of network traffic when visiting SecureDrop; we then chose 40 target `.onion` sites which our server-side defense would morph SecureDrop's traffic to look like.

We considered a powerful adversary, who knows all sites that the defense would like to morph traffic to look like. For each target site, the adversary could train on all the undefended SecureDrop network traffic and the network traffic of the target `.onion` site, and they must classify an unknown traffic instance as either SecureDrop or the target `.onion` site. In our experiment, all new traffic instances were the morphed SecureDrop page; under a perfect defense all should have been classified as the target site

Using k-FP with 1,000 trees [11], the average binary classification accuracy over the 40 different `.onion` sites was 0.372 ± 0.416. Overall, our server-side defense was successful in obscuring which site a client was visiting, though we saw a large variation: some onion sites perfectly concealed the true label while others failed.

The average communication cost (incoming and outgoing size of packets) of the SecureDrop page was 15 KB, and it loaded on average in 4.62 seconds. The average communication cost of the morphed page was 373 KB and it loaded in

6.70 seconds. The size of the morphed page entirely depends on the target page we chose to morph the SecureDrop page towards, if a smaller target page had been chosen this would result in a smaller bandwidth overhead. However, the average bandwidth overhead is still smaller than that of a standard website.

Facebook

To generalize our defense beyond SecureDrop we chose 100 `.onion` sites that may also wish to protect visiting clients from WF attacks, by morphing their traffic to that of the Facebook `.onion` site[17]. We collected 40 traffic instances for each `.onion` site. All WF attacks were applied in the same manner as in subsection 6.1.

Binary classification: the average binary classification accuracy over the 100 `.onion` sites was 0.098 ± 0.253. Even when the adversary knows undefended and target site, the attack's accuracy is below 10%.

Closed World classification: we also compared a closed world attack on the 100 undefended `.onion` sites and the same attack after morphing those sites to look like Facebook `.onion` site. If our server side defense is successful the 100 morphed `.onion` sites should, at the network level, look like the Facebook `.onion` site, resulting in a low classification accuracy.

Table 8, shows as expected, attack accuracy decreases when onion sites are morphed to resemble Facebook's network traffic patterns.

Table 7: Facebook experiment latency and bandwidth overheads.

	Latency		Volume	
	%	Avg. (s)	%	Avg. (KB)
Undefended	–	3.99	–	175
Defended	27.3	5.08	80	315

Open World classification: in addition to closed world experiments, we evaluated the server-side defense in the open world setting, where we included instances of `.onion` sites that were not of interest to the attacker. We used

[17] `https://facebookcorewwwi.onion`

Table 8: Closed world classification for `.onion` sites morphed to Facebook's `.onion` site.

	k-NN (%)	k-FP (%)	CUMUL (%)
Undefended	45.6	69.6	55.6
Defended	9.4	55.6	53.6

5,259 unique `.onion` sites, from subsection 5.2, as background traffic instances. Table 9 shows, as expected, attack's accuracy decreases when sites are morphed to resemble Facebook's network traffic patterns.

Table 9: Open world classification for `.onion` sites morphed to Facebook's `.onion` site.

	k-NN (%)		k-FP (%)		CUMUL-k-FP (%)	
	TPR	FPR	TPR	FPR	TPR	FPR
Undefended	30.8	2.6	59.3	5.2	53.2	5.7
Defended	7.8	0.9	44.9	1.8	44.4	2.0

Table 7 shows the average time to load a page only increases by 1.09s when morphing a page to the Facebook `.onion` site. We also see that the bandwidth overhead is, compared to previous works, quite tolerable. The total cost of communication rises by only 140KB.

B KDE distributions

We used Kernel Density Estimation (KDE) to estimate the distributions of number of objects (Figure 7), size of html pages (Figure 8) and size of objects (Figure 9). KDE is a non-parametric method for estimating a probability distribution given a data sample, which provides smoother estimates than histograms. KDE requires to specify a kernel (Gaussian, in our case) and a bandwidth. The bandwidth impacts on the smoothness of the estimate: a larger bandwidth tends to provide better smoothness, but less fidelity to the

original data. To determine the bandwidth for each of our distributions, we first performed Grid Search Cross Validation using `scikit-learn` library[18], to obtain a rough idea of the bandwidth ranges. Then, we manually trimmed the bandwidth to achieve what visually seemed to reflect well the variance of data, but also provided smooth distributions. For our purposes, it was important to have smooth estimates to guarantee a good quality in sampling (e.g., to avoid spikes). We used a bandwidth of 2 for the distribution over objects, and of 2000 for both the HTML and object sizes distributions.

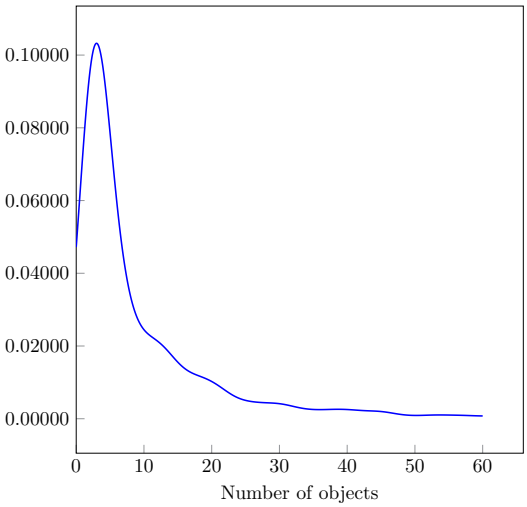


Figure 7: KDE distribution of the number of objects.

[18] `http://scikit-learn.org/`

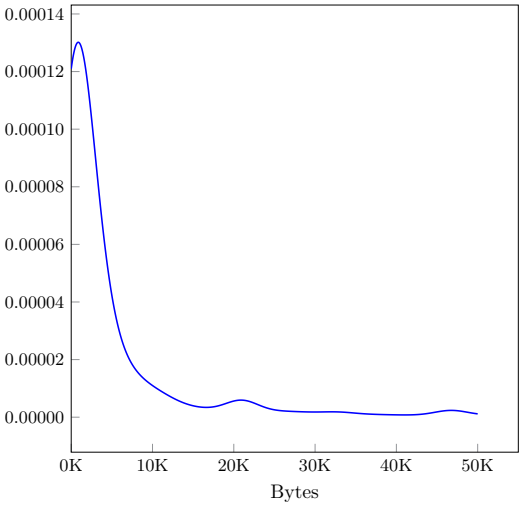


Figure 8: KDE distribution of the HTML sizes.

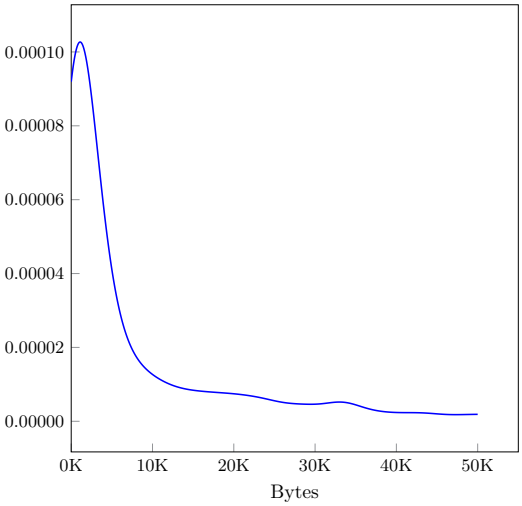


Figure 9: KDE distribution of the object sizes.

Publication

How Unique is your Onion? An Analysis of the Fingerprintability of Tor Onion Services

Publication Data

Overdorf, R., Juarez, M., Acar, G., Greenstadt, R., and Diaz, C. How unique is your onion? an analysis of the fingerprintability of tor onion services. In *ACM Conference on Computer and Communications Security (CCS)* (2017), ACM, pp. 2021–2036

Contributions

- Secondary author with substantial contributions.

How Unique Is Your Onion? An Analysis of the Fingerprintability of Tor Onion Services

Rebekah Overdorf[1], Marc Juarez[2], Gunes Acar[2], Rachel Greenstadt[1], and Claudia Diaz[2]

[1] Drexel University, Philadelphia, US
[2] KU Leuven, ESAT/COSIC and iMinds, Leuven, Belgium

Abstract. Recent studies have shown that Tor onion (hidden) service websites are particularly vulnerable to website fingerprinting attacks due to their limited number and sensitive nature. In this work we present a multi-level feature analysis of onion site fingerprintability, considering three state-of-the-art website fingerprinting methods and 482 Tor onion services, making this the largest analysis of this kind completed on onion services to date.

Prior studies typically report average performance results for a given website fingerprinting method or countermeasure. We investigate which sites are more or less vulnerable to fingerprinting and which features make them so. We find that there is a high variability in the frequency at which sites are classified (and misclassified) by these attacks, implying that average performance may not be informative of the risks of particular sites.

We use a number of methods to analyze the features exploited by the different website fingerprinting methods and discuss what makes onion service sites more or less easily identifiable, both in terms of their traffic traces as well as their webpage design. We study misclassifications to understand how onion services sites can be redesigned to be less vulnerable to website fingerprinting attacks. Our results also inform the design of website fingerprinting countermeasures and their evaluation considering likely disparate impact across sites.

1 Introduction

Website fingerprinting attacks apply supervised classifiers to network traffic traces to identify patterns that are unique to a web page. These attacks can

circumvent the protection afforded by encryption [7,13,19,25] and the metadata protection of anonymity systems such as Tor [9,12]. To carry out the attack the adversary first visits the websites, records the network traffic of his own visits, and extracts from it a *template* or *fingerprint* for each site. Later, when the victim user connects to the site (possibly through Tor), the adversary observes the victim's traffic and compares it to the previously recorded templates, trying to find a match. Website fingerprinting can be deployed by adversaries with modest resources who have access to the communications between the user and the Tor entry guard. There are many entities in a position to access this communication, including wireless router owners, local network administrators or eavesdroppers, Internet Service Providers (ISPs), and Autonomous Systems (ASes), among other network intermediaries.

Despite the high success rates initially reported by website fingerprinting attacks [6,28], their practicality in the real-world remains uncertain. A 2014 study showed that the success of the attacks is significantly lower in realistic scenarios than what is reported by evaluations done under artificial laboratory conditions [15]. Moreover, using a very large world of websites, Panchenko et al. showed that website fingerprinting attacks do not scale to the size of the Web [21], meaning that, in practice, it is very hard for an adversary to use this attack to recover the browsing history of a Tor user.

Kwon et al. demonstrated, however, that a website fingerprinting adversary can reliably distinguish onion service connections from other Tor connections [17]. This substantially reduces the number of sites to consider when only targeting onion services, as the universe of onion services is orders of magnitude smaller than the web, which makes website fingerprinting attacks potentially effective in practice. In addition, Onion services are used to host sensitive content such as whistleblowing platforms and activist blogs, making website fingerprinting attacks on this sites particularly attractive, and potentially very damaging [8]. For these reasons, we focus our analysis on onion services rather than the whole web.

In this work we choose to model the set of onion services as a closed world. Our dataset contains as many landing pages of the hidden service world as was possible for us to collect at the time. After removing pages with errors and pages that are duplicates of other sites, we were left with a sanitized dataset of 482 out of the 1,363 onion services that were crawled. While the exact size of the complete onion service world cannot be known with certainty, `onionscan` was able to find 4,400 onion services on their latest scan (this number is not sanitized for faulty or duplicated sites) [18]. This indicates that our set, while incomplete, contains a significant portion of the onion service world. We consider that an actual attacker can compile an exhaustive list of onion services, which would effectively yield a closed world scenario, since, once the adversary establishes that a user is visiting a onion service, the onion service in question will be one

on the adversary's list. We note that closed world models are not realistic when considering the entire web, rather than just onion services.

Prior evaluations of website fingerprinting attacks report aggregate metrics such as average classifier accuracy. However, we find that some websites have significantly more distinctive fingerprints than others across classifiers, and that average metrics such as overall classifier accuracy do not capture this diversity.

In this work, we study what we call the *fingerprintability* of websites and investigate what makes a page more vulnerable to website fingerprinting. This issue has practical relevance because adversaries interested in identifying visits to a particularly sensitive site may not care about the accuracy of the classifier for other sites, and thus the fingerprintability of that specific site matters. Similarly, the administrators of onion services likely care more about the vulnerability of *their* users to fingerprinting attacks, rather than the average vulnerability of a onion services to the attack. We extract lessons from our analysis to provide recommendations to onion service designers to better protect their sites against website fingerprinting attacks, including an analysis of a high profile SecureDrop instance.

The contributions of this study are:

Large .onion study. [3] We collected the largest dataset of onion services for website fingerprinting to date and evaluated the performance of three state-of-the-art classifiers in successfully identifying onion service sites. For comparison, previous studies considered worlds of 30 [11] or 50 [8, 17] onion services, an order of magnitude smaller than our study, that analyses 482 onion services.

Fingerprintability matters. While the average accuracy achieved by the classifiers is 80%, we found that some sites are consistently misclassified by *all* of the methods tested in this work, while others are consistently identified correctly, and yet others provide mixed results. In particular, 47% of sites in our data set are classified with greater than 95% accuracy, while 16% of sites were classified with less than 50% accuracy. Throughout this paper, we use the term *fingerprintable* to mean how many of the visits are correctly classified. Depending on the requirements of the specific analysis, we use different ways to distinguish more and less fingerprintable sites. This includes comparing top 50 sites to bottom 50 sites or taking sites with $F1 < 0.33$ as less fingerprintable and sites with $F1 > 0.66$ as more fingerprintable.

Errors made by different methods are correlated. Fully 31% of misclassified instances were misclassified by all three classifiers. This implies that weaknesses of the individual classifiers cannot be fully overcome using

[3] This data along with the code used for analysis in this work is available at `https://cosic.esat.kuleuven.be/fingerprintability/`

ensemble methods. We nonetheless propose an *ensemble* that combines all three classifiers, slightly improving the results offered by the best individual classifier.

Novel feature analysis method. We present a method for analyzing fingerprintability that considers the relationship between the inter-class variance and intra-class variance of features across sites. The results of this analysis explain which features make a site fingerprintable, independently of the classifier used.

Size matters. We show that size-based features are the most important in identifying websites and that when sites are misclassified, they are typically confused with sites of comparable size. We show that large sites are consistently classified with high accuracy.

Dynamism matters for small sites. While large sites are very fingerprintable, some small sites are harder than others to classify. We find that misclassified small sites tend to have more variance, and that features related to size variability are more distinguishing in sets of small sites. Put simply, smaller sites that change the most between visits are the hardest to identify.

Analysis of site-level features. Site-level features are website design features that cannot be (directly) observed in the encrypted stream of traffic but can be tweaked by the onion service operators. We identify which site-level features influence fingerprintability and we provide insights into how onion services can be made more robust against website fingerprinting attacks.

Insights for Adversarial Learning. Website fingerprinting is a dynamic, adversarial learning problem in which the attacker aims to classify a traffic trace and the defender aims to camouflage it, by inducing misclassifications or poisoning the learning system. In the parlance of adversarial learning [2], we have conducted an exploratory attack against three different approaches, to help site owners and the Tor network design better causative attacks. A causative attack is an attack against a machine learning system that manipulates the training data of a classifier. Most adversarial learning approaches in the literature consider the adversary to be the evader of the learning system, not the learner. However, this is not the case in website fingerprinting nor in many other privacy problems. For this reason, most adversarial learning studies investigate an attack on a specific learning algorithm and feature set. In contrast, we study the three top-performing learners and introduce a classifier-independent feature analysis method to study the learnability of a particular class (a web page).

2 Background and related work

Encryption alone does not hide source and destination IP addresses, which can reveal the identities of the users and visited website. Anonymous communications systems such as Tor [9] route communications through multiple relays, concealing the destination server's address from network adversaries. Moreover, Tor supports onion services which can be reached through Tor while concealing the location and network address of the server.

Website fingerprinting is a traffic analysis attack that allows an attacker to recover the browsing history of a user from encrypted and anonymized streams. Prior work has studied the effectiveness of this attack on HTTPS [7], encrypted web proxies [13, 25], OpenSSH [19], VPNs [12], and various anonymity systems such as Tor and JAP [12]. We focus on Tor because it is, with more than two million daily users [1], the most popular anonymous communications system.

In website fingerprinting the adversary is a network eavesdropper who can identify the user by her IP address, but who does not know which website the user is visiting (see Figure 1). The attacker cannot decrypt the communication, but can record the network packets generated by the activity of the user. To guess the web page that the user has downloaded, the attacker compares the traffic recorded from the user with that of his own visits to a set of websites. The best match is found using a statistical classifier.

Website fingerprinting attacks are based on supervised classifiers where the training instances are constructed from the traffic *samples* or *traces* the adversary collects browsing sites of interest with with Tor, and the test samples are traces presumably captured from Tor users' traffic. Next, we will give an overview of website fingerprinting attacks that have been proposed in the literature.

2.1 Attacks against Tor

In 2009, Herrmann et al. proposed the first website fingerprinting attack against Tor, based on a Naive Bayes classifier and frequency distributions of packet lengths [12]. Their study only achieved an average accuracy of 3% for 775 websites, but their attack was improved by Panchenko et al. who used a Support Vector Machine (SVM) and extracted additional features from traffic bursts to classify Herrmann et al.'s dataset with more than 50% accuracy [22].

Panchenko et al.'s study was also the first to perform an open-world evaluation of website fingerprinting attacks [22]. Prior work relied on a *closed-world assumption*, which assumes that the universe of possible pages is small enough that adversary can train the classifier on all sites. The open-world evaluation is

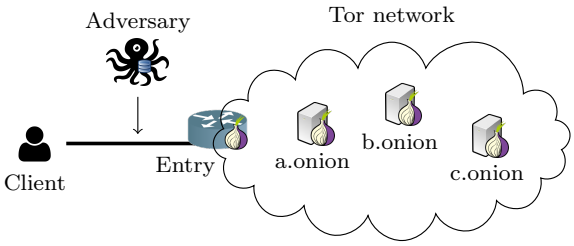


Figure 1: The client visits an onion service site over the Tor network. The adversary has access to the (encrypted) link between the client and the *entry* to the Tor network. For clarity, we have omitted the six-hop circuit between the client and the onion service. The attacker cannot observe traffic beyond the entry node.

appropriate for a web environment as it accounts for users visiting pages that the classifier has not been trained on. Based on Herrman et al.'s dataset, Cai et al. [6] achieved more than 70% accuracy in an open world setting. Wang and Goldberg's [28] approach obtained over 90% accuracy for 1,000 sites in an open world setting.

The results reported by these attacks were criticized for using experimental conditions that gave unrealistic advantages to the adversary, compared to real attack settings [15]. However, new techniques have been shown to overcome some of those limitations, suggesting that attacks may be successful in the wild [26].

Even though an open world is a more realistic evaluation setting than a closed world for the web, our evaluation considers a closed world because: i) the universe of onion services is small enough that is feasible for an adversary to build a database of fingerprints for all existing onion services; and ii) we are interested in the best-case scenario for the adversary because we evaluate the vulnerability to website fingerprinting from a defender's point of view.

As in most prior work on website fingerprinting, we only consider the homepages of the websites and not inner pages within a website. We justify this for onion services by arguing that, given their unusable naming system and their shallowness in terms of not having a deep structure, it is reasonable to assume that visitors of onion services will land first on homepage more often than for regular sites before logging in or further interacting with the site.

In this paper, we focus only on onion services because a 2015 study showed that the website fingerprinting adversary can distinguish between visits to onion services and regular websites with high accuracy [17]. Even though Panchenko et al.'s study shows that website fingerprinting does not scale to the Web, Website

fingerprinting has been identified as a potential threat for onion services for two reasons [8]: first, in contrast to the Web's size, the onion service space's size may be sufficiently small for an adversary to build a fingerprint database for all existing onion services; second, onion services tend to host sensitive content and visitors of these sites may be subject to more serious, adverse consequences.

2.2 State-of-the-art attacks

We have selected three classifiers proposed in recent prior work for our study because they represent the most advanced and effective website fingerprinting attacks to date. Each attack uses different classification algorithms and feature sets, although they have some features in common. The details of each classifier are as follows:

Wang-kNN [27]: Wang et al. proposed an attack based on a k-Nearest Neighbors (k-NN) classifier that used more than 3,000 traffic features. Some of the most relevant features are the number of outgoing packets in spans of 30 packets, the lengths of the first 20 packets, and features that capture traffic *bursts*, i.e., sequences of packets in the same direction. They also proposed an algorithm to tune the weights of the custom distance metric used by the k-NN that minimizes the distance among instances that belong to the same site. They achieved between 90% to 95% accuracy on a closed world of 100 non-onion service websites [27]. Kwon et al. evaluated their own implementation of the attack for 50 onion service sites and obtained 97% accuracy.

CUMUL [21]: Panchenko et al. designed CUMUL, an attack based on a Radial Basis Function kernel (RBF) SVM. Each feature instance is a 104-coordinate vector formed by the number of bytes and packets in each direction and 100 interpolation points of the cumulative sum of packet lengths (with direction). They report success rates that range between 90% and 93% for 100 regular sites. In addition, they collected the largest and most realistic dataset of non-onion service websites, including *inner* pages of websites and popular links extracted from Twitter. They conclude that website fingerprinting does not scale to such large dataset, as classification errors increase with the size of the world.

k-Fingerprinting (k-FP) [11]: Hayes and Danezis's k-FP attack is based on Random Forests (RF). Random Forests are ensembles of decision trees that are randomized and averaged to reduce overfitting. In the open world, they use the leafs of the random forest to encode websites. This allows them to represent websites in function of the outputs of the random forest, capturing the relative distance to pages that individual trees have confused with the input page. The

instances extracted from the random forest are then fed into a k-NN classifier for the actual classification. The study uses a set of 175 features that includes variations of features in the literature as well as timing features such as the number of packets per second. Hayes and Danezis evaluated the attack on a limited set of 30 onion services and obtained 90% classification accuracy [11].

In the following subsection we provide an overview of prior results on features that has inspired the feature selection made by these three attacks.

2.3 Feature analysis for website fingerprinting

We consider two types of features: *network-level* and *site-level* features. Network-level features are extracted from the stream of TCP packets and are the typical features used in website fingerprinting attacks. Site-level features are related to the web design of the site. These features are not available in the network traffic meta-data, but the adversary still has access to them by downloading the site.

Most website fingerprinting feature analyses have focused on network-level features and have evaluated their relevance for a specific classifier [5, 10, 22]. In particular, Hayes and Danezis [11] perform an extensive feature analysis by compiling a comprehensive list of features from the website fingerprinting literature as well as designing new features. In order to evaluate the importance of a feature and rank it, they used the random forest classifier on which their attack is based.

Unlike prior work, our network-level feature analysis is classifier-independent, as we measure the statistical variance of features among instances of the same website (*intra-class variance*) and among instances of different websites (*inter-class variance*).

2.4 Website fingerprinting defenses

Dyer et al. presented *BuFLO*, a defense that delays real messages and adds dummy messages to make the traffic look constant-rate, thus concealing the features that website fingerprinting attacks exploit. They conclude that coarse-grained features such as page load duration and total size are expensive to hide with BuFLO and can still be used to distinguish websites [10].

There have been attempts to improve BuFLO and optimize the padding at the end of the page download to hide the total size of the page [4, 6]. These defenses however incur high latency overheads that make them unsuitable for Tor. To

avoid introducing delays, a website fingerprinting defense based solely on adding dummy messages was proposed by Juarez et al. [16]. These defenses aim at crafting padding to obfuscate distinguishing features exploited by the attack. Instead, we look at sites and examine what makes them more or less fingerprintable.

There are defenses specifically designed for Tor that operate at the application layer [8, 20, 23]. However, these defenses do not account either for feature analyses that can help optimize the defense strategy. Our study is the first to analyze the features at both the website and network layers. Based on our results, we discuss ways to reduce the fingerprintability of onion service sites and inform the design of server and client-side website fingerprinting defenses without requiring any changes to the Tor protocol itself.

3 Data collection and processing

We used the onion service list offered by `ahmia.fi`, a search engine that indexes onion services. We first downloaded a list of 1,363 onion service websites and found that only 790 of them were online using a shell script based on `torsocks`. We crawled the homepage of the 790 online onion services.

Prior research on website fingerprinting collected traffic data by grouping visits to pages into batches, visiting every page a number of times each batch [15, 28]. All visits in a batch used the same Tor instance but Tor was restarted and its profile wiped between batches, so that visits from different batches would never use the same circuit. The batches were used as cross-validation folds in the evaluation of the classifier, as having instances collected under the same circuit in both training and test sets gives an unfair advantage to the attacker [15, 28].

In this study, we used the same methodology to collect data, except that we restarted Tor on every visit to avoid using the same circuit to download the same page multiple times. We ran the crawl on a cloud based Linux machine from a data center in the US in July 2016. The crawl took 14 days to complete which allowed us to take several snapshots of each onion service in time.

We used Tor Browser version 6.0.1 in combination with Selenium browser automation library [4]. For each visit, we collected network traffic, HTML source code of the landing page, and HTTP request-response headers. We also saved a screenshot of each page.

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

We captured the network traffic traces using the `dumpcap` [5] command line tool. After each visit, we filtered out packets that were not destined to the Tor guard node IP addresses. Before each visit, we downloaded and processed the Tor network consensus with `Stem` [6] to get the list of current guard IP addresses.

The HTML source code of the index page was retrieved using Selenium's `page_source` property. The source code and screenshots are used to extract site-level features, detect connection errors and duplicate sites. The HTTP requests and response headers are stored using a custom Firefox browser add-on. The add-on intercepted all HTTP requests, including the dynamically generated ones, using the `nsIObserverService` of Firefox [7].

Finally, we collected the logs generated by Tor Browser binary and Tor controller logs by redirecting Tor Browser's process output to a log file.

3.1 Processing crawl data

We ran several post-processing scripts to make sure the crawl data was useful for analysis.

Remove offline sites. Analyzing the collected crawl data, we removed 573 sites as they were found to be offline during the crawl.

Remove failed visits. We have also removed 14481 visits that failed due to connection errors, possibly because some onion sites have intermittent uptime and are reachable temporarily.

Outlier removal. We used Panchenko et al.'s outlier removal strategy to exclude packet captures of uncommon sizes compared to other visits to the same site [21]. This resulted in the removal of 5264 visits.

Duplicate removal. By comparing page title, screenshot and source code of different onion services, we found that some onion service websites are served on multiple `.onion` addresses. We eliminated 159 duplicate sites by removing all copies of the site but one.

Threshold by instances per website. After removing outliers and errored visits, we had an unequal number of instances across different websites. Since the number of training instances can affect classifier accuracy, we set all websites to have the same number of instances. Most datasets in the literature have between 40 and 100 instances per website and several evaluations have shown

[5] `https://www.wireshark.org/docs/man-pages/dumpcap.html`
[6] `https://stem.torproject.org/`
[7] `https://developer.mozilla.org/en/docs/Observer_Notifications#HTTP_requests`

that the accuracy saturates after 40 instances [21, 28]. We set the threshold at 70 instances which is within the range of number of instances used in the prior work. Choosing a greater number of instances would dramatically decrease the final number of websites in the dataset. We removed 84 sites for not having a sufficient number of instances and removed 9,344 extra instances.

Feature Extraction. Following the data sanitization steps outlined above, we extract features used by the three classifiers. Further, we extract *site level* features using the HTML source, screenshot, HTTP requests and responses. Site level features are explained in Section 6.

In the end, the dataset we used had 70 instances for 482 different onion sites.

4 Analysis of website classification errors

This section presents an in-depth analysis of the successes and failures of the three state-of-the-art website fingerprinting methods. This analysis helps identify which pages are the most fingerprintable and which are more likely to confuse the classifiers, giving insight into the nature of the errors produced by the classifiers.

4.1 Classifier accuracy

Even though the classification problem is not binary, we binarize the problem by using a *one-vs-rest* binary problem for each site: a True Positive (TP) is an instance that has been correctly classified and False Positive (FP) and False Negative (FN) are both errors with respect to a fixed site w; a FP is an instance of another site that has been classified as w; a FN is an instance of w that has been classified as another site.

In the closed world we measure the accuracy using the F1-Score (F1). The F1-Score is a *complete* accuracy measure because it takes into account both Recall (TPR) and Precision (PPV). More precisely, the F1-Score is the harmonic mean of Precision and Recall: if either is zero, the F1-Score is zero as well, and only when both achieve their maximum value, the F1-Score does so too.

Note that there are the same *total* number of FPs and FNs, since a FP of w_y that actually belongs to w_x is at the same time a FN of w_x. Thus, in the closed world the total F1-Score equals both Precision and Recall. However, when we focus on a particular site, the FP and FN for that site are not necessarily the same (see Table 2).

Table 1: Closed world classification results for our dataset of 482 onion services (33,740 instances in total).

	k-NN	CUMUL	k-FP
TPR	69.97%	80.73%	77.71%
FPR	30.03%	19.27%	22.29%

We have applied the classifiers to our dataset of 482 onion services and evaluated the classification using 10-fold cross-validation. Cross-validation is a standard statistical method to evaluate whether the classifier generalizes for instances that it has not been trained on. In most cases, ten is the recommended number of folds in the machine learning literature and the standard in prior website fingerprinting work. The results for each classifier are summarized in Table 1 where we report the total number of TPs and FPs and the average accuracy obtained in the 10-fold cross-validation. Thus, we note that using TPR as an accuracy metric is sound in the closed world but, in the open world, TPR is a partial measure of accuracy, as it does not take into account Precision.

As we see in Table 1, while CUMUL and k-FP achieve similar accuracies, the k-NN-based attack is the least accurate. Even though these results are in line with other studies on website fingerprinting for onion services [8], we found some discrepancies with other evaluations in the literature. For 50 sites, Hayes and Danezis obtain over 90% accuracy with k-FP [11], and Kwon et al. obtained 97% with k-NN [17]. However, for the same number of sites and even more instances per site, our evaluations of k-FP and k-NN only achieve 80% maximum accuracy. Since our results show that some sites are more fingerprintable than others, we believe the particular choice of websites may account for this difference: we randomly picked 50 sites from our set of 482 sites and even though Kwon et al. also used onion URLs from ahmia.fi, they do not explain how they picked the URLs for their evaluation.

4.2 Classifier variance

In order to determine which features cause a site to be fingerprintable, we look into two types of sites: i) sites that are easy to fingerprint, i.e., sites that consistently cause the least amount of errors across all classifiers; and ii) sites that are difficult to fingerprint, namely sites that are most frequently misclassified across all three classifiers. In the following sections, we compare

the features of these two types of sites and look for evidence that explains their different degree of fingerprintability.

In our analysis, we evaluated the accuracy for each website in isolation and ranked all the websites to find a threshold that divides them into the two types described above. We found that only 10 (in kNN) to 40 (in CUMUL) sites are perfectly classified, while the other sites have at least one misclassified instance – some of them are consistently misclassified by all three classifiers.

Table 2: The top five onion services by number of misclassification for each attack (repeating services in bold).

	URL (.onion)	TP	FP	FN	F1
k-NN	4fouc...	4	84	66	0.05
	ykrxn...	3	62	67	0.04
	wiki5k...	3	77	67	0.04
	ezxjj...	2	76	68	0.03
	newsi...	1	87	69	0.01
CUMUL	zehli...	2	15	68	0.05
	4ewrw...	2	29	68	0.04
	harry...	2	29	68	0.04
	sqtlu...	2	35	68	0.04
	yiy4k...	1	14	69	0.02
k-FP	**ykrxn**...	4	62	66	0.06
	t4is3...	3	42	67	0.05
	wiki5...	3	55	67	0.05
	jq77m...	2	54	68	0.03
	newsi...	2	63	68	0.03

We have compared the misclassifications of all three attacks to find sites that are misclassified by all the classifiers as opposed to sites that at least one of identified correctly. Table 2 shows the top five onion services ranked by number of misclassifications, where we see a partial overlap of which sites are misclassified the most. This means there is not only variation across websites within a given classifier but also across different classifiers.

4.3 Comparison of website classification errors

Figure 2 shows a scaled Venn diagram of the classification errors. The circles represent the errors made by each of the classifiers, and the intersections

represent the fraction of instances misclassified by the overlapping classifiers. All numbers in the Venn diagram add to one as each number is a fraction of all misclassifications, not a fraction of the misclassifications for a specific classifier. This is to represent how misclassifications are distributed over classifiers and intersections of classifiers. The black region in the center represents the errors that are common to all three classifiers, which accounts for 31% of all classification errors. This large intersection indicates that classification errors for a given website are correlated and not independent for each classifier. Note that if the errors were independent, the adversary would benefit from employing multiple website fingerprinting classifiers; but the correlation suggests that such gains will have limited returns.

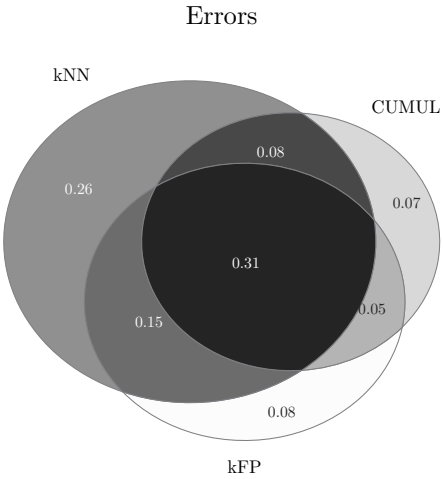


Figure 2: Scaled Venn diagram of classification errors. Each circle represents the set of prediction errors for a method: *kNN*, *CUMUL* and *kFP*. In the intersections of these circles are the instances that were incorrectly classified by the overlapping methods. 31% of the erred instances were misclassified by all three methods, suggesting strong correlation in the errors.

The diagram in Figure 2 does not take into account whether the classifiers that erred predicted the same mistaken label or not. In Figure 3, we depict the Venn diagram of misclassifications according to the (erroneous) guessed label. The percentage of instances that were mislabeled in the same way by all three classifiers is substantially smaller: only 2% of the errors are errors that all three classifiers erred with the same predicted label. Interestingly, this small intersection implies that even though these classifiers err on the same instances (Figure 3), they do so in different ways, making different predictions for a given instance.

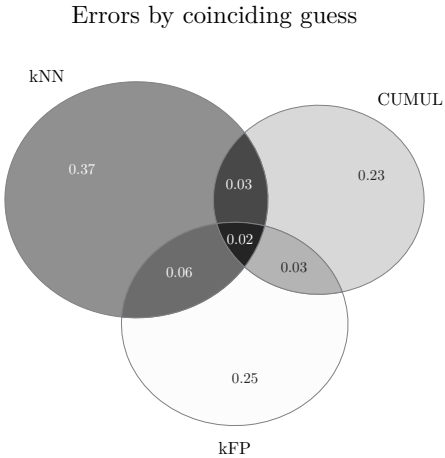


Figure 3: Scaled Venn diagram of classifications errors by coinciding guess. The intersections contain instances that were incorrectly classified with *exactly* the same label by the overlapping classifiers. Only 2% of the errors were misclassified to the same incorrect site by all three methods, while 85% were misclassified differently by each method, showing that the methods do err in different ways.

4.4 Ensemble classifier

In Figure 2 we observe that more than 25% of the errors occur in only one of the methods, and an additional 17% of errors appear in only two of the methods. A third of the errors were misclassified by all three methods. Thus, an *ensemble* classifier that appropriately combines the three classifiers can achieve higher accuracy than any individual classifier alone, by correcting classification errors that do not occur in all the methods.

We can estimate the maximum improvement that such an ensemble could achieve by looking at the potential improvement of the best classifier. In our case, CUMUL has the greatest accuracy with 874 errors that could be corrected using kNN or kFP. So if CUMUL did not make these errors, its accuracy would be improved by $\frac{874}{33{,}740} = 2.6\%$. Even though the margin for improvement is small, we build an ensemble to reduce the dependency of our results on a single classifier. In addition, by choosing an ensemble we ensure that we are not underestimating an adversary that combines all the state-of-the-art classifiers. We therefore use the results of the ensemble to determine fingerprintability, and compute a site's ***fingerprintability score*** **as its F1 score from the ensemble classifier.**

We analyze the overlap in errors and TPs for the three classifiers for different ensemble methods, as follows:

Random. For each instance, randomly select one of the predictions of the three classifiers. With this method the ensemble achieves 79.98% accuracy.

Highest confidence. For each instance, take the prediction of the classifier with highest confidence. kFP and CUMUL use Random Forests and SVM respectively, and both output a classification probability for each of the possible classes. For kNN we use the distance to the nearest neighbor as the confidence metric. The accuracy was 80.91% using this method.

$\mathbf{P_1 - P_2}$ Diff. For each instance, use the output of the classifier with the greatest difference in confidence between its first and second predictions. We obtained 80.91% accuracy with this method.

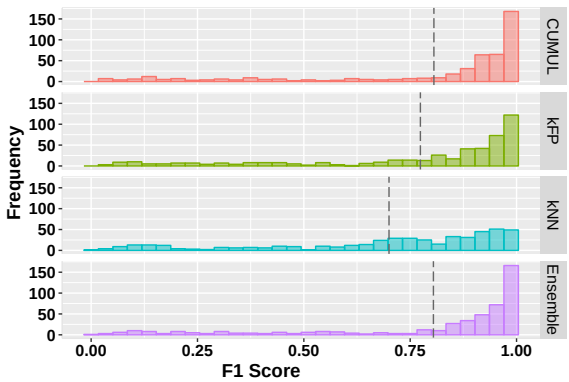


Figure 4: F1 score histograms for each classifier. Vertical dashed lines represent the mean F1 score.

We decided to use the $P_1 - P_2$ Diff for the rest of our analysis because it uses most information about the confidence vector. Figure 4 shows the F1 score histograms for all classifiers including the ensemble. The vertical dashed lines show the mean F1-scores. We note that the ensemble is only marginally better than CUMUL. The main visible difference is in the relative weights of the second and third highest bars: the ensemble improves the F1 score for a subset of instances that in CUMUL contribute to the third bar, and to the second in the ensemble.

In the histograms we can once more see the accuracy variation across sites (horizontally) and across classifiers (vertically). Even though for CUMUL and the ensemble most of the sites have high F1 scores, we see there still are several

sites in the low ranges of F1 scores that even CUMUL and ensemble cannot perfectly fingerprint (the ones shown in Table 2).

4.5 Sources of classification error

In order to gain insight about the nature of the classifier errors, we performed an exploratory analysis specific to the features of the erred instances. We use the *total incoming packet size* as example for illustrating the analysis, because, as we show in the following sections, it is the most salient feature. However, this analysis can as well be applied to any other feature.

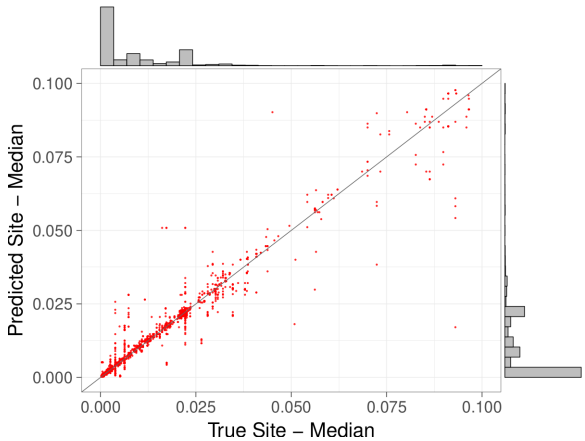


Figure 5: Median of total incoming packet size for misclassified instances (true vs predicted site). We also plot the dashed diagonal line, $y = x$, for comparison. We chose the total incoming packet size for this analysis because it is the most distinguishing feature (see Section 5).

In Figure 5, each point represents a misclassified instance, with the x axis value being the median incoming packet size of the 'true site' (site the instance truly belongs to), and the y axis value being the median incoming packet size of the 'predicted site' (according to the *ensemble* classifier). Note that the total incoming packet sizes have been normalized to the interval $[0, 1]$ using Min-Max normalization across all instances. For visualization purposes, we have clipped the range to focus on the region where approximately 80% of the data points are (101 points were excluded).

Figure 5 shows that the median incoming packet sizes of the predicted and true sites are highly correlated: most of the instances are close to the diagonal $y = x$ (dashed line), meaning that for most of the errors, true and predicted

sites are similar to each other in terms of median incoming packet size. In fact, since the median incoming packet size approximates to the median total size of the page, this shows that most of the misclassified pages were confused with pages of similar size. Furthermore, as shown by the histograms most of the misclassifications occur on pages of small sizes, confirming the hypothesis that large pages are easier to identify.

We also measure the deviation of each instance from its class mean. We use *Z-score*, which indicates the number of standard deviations a sample is away from the mean. The Z-score is a standard statistic that normalizes the deviation from the mean using the class' standard deviation. Unlike the standard deviation, this allows to compare Z-scores between classes with standard deviations that differ by orders of magnitude. This property is suited to our case because the sites in our set have large differences in terms of the total incoming packet sizes.

On the left side of Figure 6 we plot the density for the deviation from the median for the total incoming packet size feature. Z-score values around the origin correspond to low-deviation, whereas values far from the origin correspond to high-deviation. We observe that the correctly classified instances are more concentrated in the center, while the misclassified instances are more concentrated in the extremes. This confirms that the instances with higher deviation from their class mean are more likely to be misclassified.

The right subfigure in Figure 6 shows the number of correctly and erroneously classified instances for the $1,755$ outliers found in our dataset. We used the Tukey's method for outlier removal based on the inter-quartile range and the first and third quartiles to identify outliers. The bar plot shows that an outlier is three times more likely to be misclassified $(1,327)$ than correctly classified (428). An instance is counted as misclassified if it is misclassified by at least one of the classifiers.

Figure 6 suggests that variation within a class such as that produced by web page dynamism can be beneficial to induce confusions with other pages.

4.6 Confusion graph

Confusion matrices have been used in prior website fingerprinting literature to visualize and help understand the nature of confusions [11, 21]. However, for a multi-class problem of size 482, the confusion matrix is too large for any visualization to be useful. This can be addressed by using *confusion graphs* instead, which represent misclassifications as a directed graph [29].

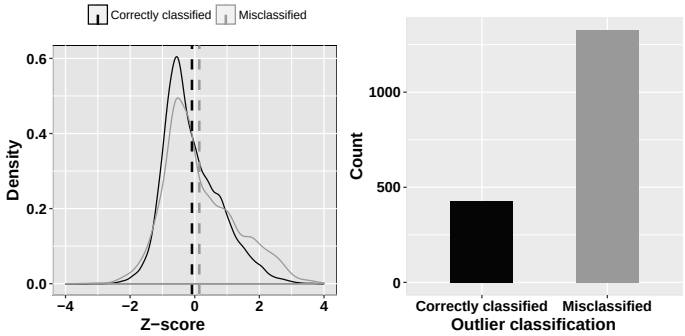


Figure 6: Density plot for absolute value of Z-score distribution of total incoming packet size. Correctly classified (dark gray) and misclassified (light gray) instances are plotted separately to contrast them with respect to their deviation from the class mean.

To better understand the nature of classification errors we draw a directed graph where nodes represent classes (onion services) and edges represent misclassifications. Source and target nodes of an edge represent true and predicted sites, respectively. The edge weight encodes the misclassification frequency (i.e., number of times the source class is misclassified as the target class). We have created a confusion graph for CUMUL, which is the best performing classifier in our dataset, shown in Figure 10 in the Appendix.

The nodes are colored based on the community they belong to, which is determined by the Louvain community detection algorithm [3], as implemented in the Gephi graph software. Node size is drawn proportional to the node degree. We observe highly connected communities on the top left, and the right which suggests clusters of onion services which are commonly confused as each other. Further, we notice several node pairs that are commonly classified as each other, forming *ellipses*.

The mean *outdegree* and *indegree* of the graph is 4.9, meaning that, on average, a site is misclassified as 5 distinct sites and confused with 5 distinct sites. The onion service with the maximum outdegree had 42 outgoing edges, meaning it is misclassified as 42 distinct sites. The onion service with the maximum indegree had 28 incoming edges, meaning it is confused with as many different sites. Interestingly, the same onion service has zero outdegree, i.e., its instances are never misclassified as belonging to another site.

We have looked into the size of the sites for each community in the graph. The sites in the dark green community at the bottom of the graph are all of

similar size and significantly larger than all the others, explaining why they are confused between each other and clustered into a community. For the other communities, however, it is not obvious which common features define the community. Further, we discovered that a few of the pairs of sites that form ellipses are false negatives of our duplicates detection in the data cleansing step, while the others require further analysis. We leave a more detailed graph-based analysis of these communities for future work.

We analyze three cases of the symmetry of classifications:

- Symmetrical: Site A is misclassified as other sites and other sites are misclassified as Site A.
- Asymmetrical: One or more sites are misclassified as Site A, but A is consistently classified as A.
- Asymmetrical: Site A is misclassified as one or more other sites, but other sites are rarely misclassified as A.

For each distinct misclassification pair $(A \rightarrow B)$ we check whether there is a symmetric misclassification $(B \rightarrow A)$. The total number of misclassifications with symmetric counterparts:

- CUMUL: 74.8% (4868/6502)
- kFP: 73,4% (5517/7519)
- kNN: 80.6% (8174/10132)

The results show the majority of the misclassifications are symmetrical, meaning that there are sets of pages that provide cover for each other, effectively forming anonymity sets . This suggests that onion services may benefit from designing their site to have features that enable them to join one of those sets.

5 Network-level feature analysis

We use classifier-independent feature analysis methods to determine which features are better predictors for website fingerprinting. Knowing which features are more distinct across classes and less distinct within a class helps us understand which features are important to each website fingerprinting method.

5.1 Methodology

To analyze the nature of the classification errors we borrow two concepts from the field of machine learning: *inter-* and *intra-class* (or *cluster*) variance. In particular, we use these concepts in the following sense:

The **intra-class variance** of a feature is defined as the variance of its distribution for a certain class, in this case a site. It quantifies how much the feature varies among instances of the class. In website fingerprinting, low intra-class variance indicates a feature remains stable across different visits to the same page.

Inter-class variance is a measure of how much a feature varies across different classes. We define it as the variance of the averages of the feature for each class. That is, we create a vector where each coordinate aggregates the instances of visits to a site by averaging their feature values. Then, we calculate the inter-class variance as the variance of that vector. In website fingerprinting, high-inter-class variance means that websites are very distinct from each other with respect to that feature.

In Section 4 we have shown evidence that both inter- and intra-class variance play a role as the cause of classification errors: misclassified pages have similar sizes to the pages they are confused with, and slightly larger variance in size than correctly classified ones. To rank features by taking into account both intra- and inter-class variance, we use the relative difference between the inter- and intra-class variance, where we define relative difference as: $d(x, y) = (x-y)/((x+y)/2)$. This formula normalizes the differences by their mean to values between 0 and 2, where features with a relative difference close to 0 are similar and features with a relative difference close to 2 are far apart. This allows features of different scales to be compared. We consider features that are close to 2 better predictors, as they have a relatively higher inter-class variance than intra-class variance.

Many of the features that appear as most predictive for the considered classifiers are directly related to the size of a site (e.g., the number of packets). Further, the misclassifications described in Section 4 show that the smaller sites are more likely to be misclassified. In addition to running feature analysis on the entire dataset, we also look only at the small sites to determine which other features have predictive value.

We start with an analysis of the network-level features used by the three fingerprinting attacks detailed in Section 2 and analyzed in Section 4. Most traditional applications of feature analysis aim to reduce the dimensionality of the data to more efficiently classify instances. Instead, the goal of our feature analysis is to determine which features can be modified to trick a classifier into

misclassifying an instance. Unlike many adversarial machine learning problems with the same goal, this analysis lacks knowledge of the specific classifier (or even the classification algorithm) used for fingerprinting, as there are many different classifiers in the literature to consider, and the site should ideally be hard to classify for all of them. In addition to the wide variety of classification techniques available in the current literature, novel classification techniques could be easily developed by an adversary.

Therefore, the network-level feature analysis we present here is classifier-independent. That is, we use only information about the feature values themselves and do not use classification methods to determine the importance of the features. Figure 7 shows the relationship between how likely a site is to be fingerprinted vs its size. All of the larger sites have high fingerprintability scores, while the scores of smaller sites are much more varied.

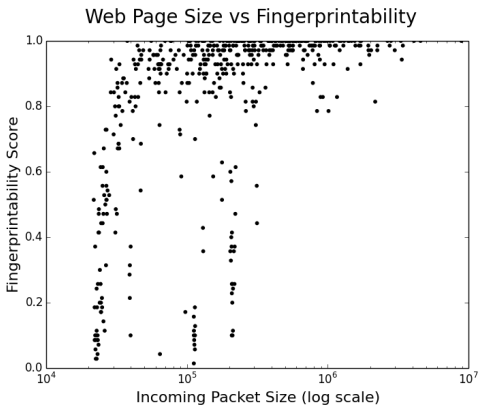


Figure 7: Larger sites are easily fingerprinted while results are mixed for smaller sites. Note also the vertical clusters of sites with low fingerprintability that are similar in size. Incoming packet size (in bytes) is plotted in log scale.

In a website fingerprinting attack, only features based on the traffic traces are available to the adversary. Each attack uses a distinct set of features derived from these traces and as a result the exact feature analysis varies.

This analysis is classifier independent, meaning no classification techniques were performed on the dataset prior to this analysis and the results do not rely on any specific classification algorithm or task. We cannot, however, perform any feature analysis that is completely independent from the website fingerprinting methods, as the types of features we analyze rely on the features chosen by each method. For each attack, however, we can determine which features are most predictive.

5.2 Network-level feature results

Here we analyze which network-level features are the best predictors in state-of-the-art website fingerprinting attacks.

CUMUL

The first group of features we consider come from the CUMUL attack. There are two types of features used in CUMUL: direct size features (Table 3) and interpolated features. The interpolated features are formed by the number of bytes and packets in each direction and 100 interpolation points of the cumulative sum of packet lengths (with direction). We calculate the inter and intra-class variance for each of these features. The direct size features are the most important to classification (Table 3). We found that the interpolated features are more predictive at the end of the trace than the beginning, with the minimum relative difference (0.37) being from the very first interpolated feature and then increasing to the greatest relative difference (1.51) being the last interpolated feature from the very end of the trace.

Table 3: Network-level feature variance analysis for the CUMUL method. These features had a higher relative difference than most of the interpolated features and alone are great predictors.

Feature Name	**Relative Diff**
Total Size of all Outgoing Packets	1.605
Total Size of Incoming Packets	1.520
Number of Incoming Packets	1.525
Number of Outgoing Packets	1.500

k-fingerprinting

The next group of features we look at come from the k-fingerprinting attack. The features used in the k-fingerprinting attack are more varied as well as more straightforward than those in CUMUL. They include not only features that give information about the size and number of packets, but also the timing of the packets. The features with the highest inter-class to intra-class variance ratio are shown in Table 4.

The feature analysis we present here is similar to the original analysis presented with the method by the authors, but without the use of any classification

technique. Further, we also look at which features are more predictive for small sites, as we see that misclassifications are much more common for smaller sites.

Table 4 shows that features correlated to the total size of a site (e.g. # of outgoing packets) have the highest relative difference and thus are among the top features. This result is consistent with the analysis done by Hayes and Danezis [11] on the same set of features.

When only smaller sites are analyzed however, standard deviation features become important. In Section 4, we show that large sites are easily identified, and the fact that size features are very predictive is not at all unexpected. However, that standard deviation features are top features for the smaller sites implies that the dynamism of the site makes a difference, as small dynamic sites are generally the least fingerprintable.

Table 4: Network-level feature analysis for kFP method.

Feature name	**Relative Diff**
All Sites	
Percent incoming vs outgoing	1.895
Average concentration of packets	1.775
# of outgoing packets	1.740
Sum of concentration of packets	1.740
Average order in	1.720
Smallest 10% of Sites	
Percent incoming vs outgoing	1.951
Average concentration of packets	1.944
Standard deviation of order in	1.934
# of packets	1.927
# of packets per second	1.927

kNN

The last set of features are those of the kNN attack. Like with the other classifiers, we find that the most important features are those that relate to the size of the traffic flow. In this case, we find that almost all of the top predictive features (with the highest relative difference) are related to "packet ordering" – which in practice acts as proxy for the size of the flow.

The packet ordering feature is computed as follows: for each outgoing packet o_i, feature f_i is the total count of all packets sent or received before it. Essentially, these features measure the ordering of incoming and outgoing packets.Note that not all sites, however, have the same number of outgoing packets. Therefore if the end of the number of outgoing packets is less than some n (we use $n = 500$ to be consistent with the original implementation), the rest of the features are filled in with zero or null values. Similarly, some sites may have over n outgoing packets. If this is the case, the packets over the n^{th} packet are ignored. Similar to the features used in CUMUL, we observed that the later features in this sequence are more important, this is because for most sites (size $< n$) they are zero and thus these features are a proxy for the total size of the site.

The only other feature-type with high relative difference between inter and intra-class variance is the number of packets (1.96), a direct measure of the size of the site.

6 Site-level feature analysis

In website fingerprinting attacks, the adversary records the network traffic between a user and Tor, and analyzes its features to identify the site that was visited. Network-level features and their relative contribution to fingerprintability are, however, not informative for onion service designers who may want to craft their site to be robust against website fingerprinting attacks. To gain insight into which design choices make sites vulnerable to attacks, and how websites can be designed with increased security, we need to look at the features at a site-level.

In this section we investigate which site-level features correlate with more and less fingerprintable sites. Site-level features are those that can be extracted from a web page itself, not from the traffic trace. Driven by adversarial learning, we investigate the task of causing misclassifications for any set of network-level features and any classification method. This information can help sites design their web pages for low fingerprintability, and also assist in developing more effective server-side defenses.

6.1 Methodology

Site-level features are extracted and stored by our data collection framework as explained in Section 3. The list of all site-level features considered can be found in Table 6 (in the Appendix).

We build a random forest regressor that classifies easy- and hard-to-fingerprint sites, using the **fingerprintability scores** (the F1 scores from the ensemble classifier described in Section 4) as labels, and considering site-level features. We then use the fingerprintability regressor as a means to determine which site-level features better predict fingerprintability.

In this section we aim to understand which site-level features are more prevalent in the most and least fingerprintable sites. For the sake of this feature analysis, we remove the middle tier of sites, defined as those with a fingerprintability score in (0.33, 0.66). 44 sites in our dataset were assigned a mid-ranged F1-score, leaving 438 sites for this analysis.

The next challenge is that the high and low-fingerprintability classes are unbalanced, because of the disproportionately higher number of easily identifiable sites compared to the amount of sites that are hard to identify. Recall that a full 47% of sites in our dataset have a fingerprintability score greater than 95%. A regressor trained with such unbalanced priors will be biased to always output a prediction for of "very fingerprintable," or values close to 1, and therefore any analysis on the results would be meaningless. To perform the feature analysis, we remove randomly selected instances from the set of more fingerprintable sites, so that it is balanced in size with that of low fingerprintability.

We train a random forest regressor using the features from Table 6. We use the feature weights from the regression to determine which of these site-level features are most predictive of sites that are easily fingerprinted. We use the information gain from the random forest regression to rank the importance of the site-level features in making websites more or less fingerprintable.

While in its current state this regression is only useful for feature analysis, this could be extended into a tool that allows sites to compute their fingerprintability score, and be able to determine if further action is needed to protect their users from website fingerprinting attacks.

6.2 Results

Figure 8 shows the results of the analysis. We see that features associated with the size of the site give the highest information gain for determining fingerprintability when all the sites are considered. Among the smallest sites, which are generally less identifiable, we see that standard deviation features are also important, implying that sites that are more dynamic are harder to fingerprint.

Additionally, Table 5 shows how different the easy- and hard-to-fingerprint sets of sites are in terms of total HTTP download size, a straightforward metric for

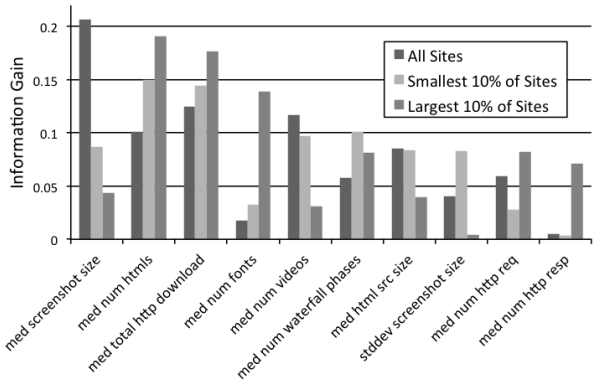


Figure 8: Most important features by information gain. Features related to the size of a site are important.

the size of a site. The median site size for the 50 most fingerprintable sites is almost 150 times larger than the median size of the harder to classify sites. The standard deviation of the total site size for the most and least fingerprintable sites, relative to their size, is similarly distinct, showing the most fingerprintable sites are less dynamic than the 50 least fingerprintable sites. That is, they are less likely to change between each visit.

Total HTTP Download Size	50 Most	50 Least
Median Std Dev	0.00062	0.04451
(normalized by total size)		
Median Size	438110	2985

Table 5: Differences in the most and least fingerprintable sites. The 50 most fingerprintable sites are larger and less dynamic than the 50 least fingerprintable sites.

While the smallest sites are less fingerprintable, some are still easily identified. Figure 9 shows the distribution of sizes considering only the smallest sites, distinguished by whether they have a high or low fingerprintability score. We can see that the least fingerprintable sites are clustered in fewer size values, while the most fingerprintable are more spread, meaning that there are fewer sites of the same size that they can be confused with.

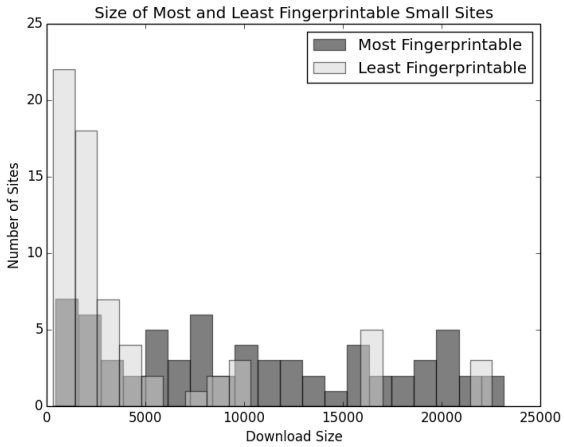


Figure 9: Distribution of sizes for the most and least fingerprintable sites, considering only the sites smaller than 25,000 bytes.

7 Implications for onion service design

Overall, our analysis showed that most onion services are highly vulnerable to website fingerprinting attacks. Additionally, we found that larger sites are more susceptible to website fingerprinting attacks. Larger sites were more likely to be perfectly classified by all attacks while many smaller sites were able to evade the same attacks by inducing misclassifications.

We also observed that the small sites that are harder to identify also have a high standard deviations for many site-level and network-level features, implying that dynamism plays a role in why these sites are less identifiable. While our results show that small size is necessary, it is not sufficient. As a result, our recommendation for onion service designers is "make it small and dynamic."

Most website fingerprinting defenses rely on some form of padding, that is, adding spurious traffic and therefore increasing the download size. Our analysis, however, shows that this type of defense may not be robust when features such as download size become sparse. Often, these defenses are tested against a single attack with a single feature set and a specific classification algorithm. We see, though, that classification errors do not always coincide for different attacks, and argue that any website fingerprinting defense needs to be tested against a

range of state-of-the-art attacks, preferably relying on different algorithms and feature sets, in order to provide more general guarantees of its effectiveness.

As a case study, we consider the results that our ensemble classifier achieved in identifying SecureDrop sites. These sites are onion services that are running the SecureDrop software, a whistleblower submission system that allows journalists and media publishers to protect the identities of their sources. Given the sensitive nature of the service that they provide and the nation-state adversaries that they may realistically face, these SecureDrop sites have strong anonymity requirements.

Our dataset contained a SecureDrop site owned by 'Project On Gov't Oversight' (POGO)[8]. The SecureDrop site had an F1-Score of 99%, meaning that it is much more vulnerable to website fingerprinting attacks than the average onion service site.

There were other SecureDrop sites present in our initial dataset, associated with The New Yorker, The Intercept and ExposeFacts. These sites were flagged as duplicates of the POGO SecureDrop site and thus removed during the data processing stage. Since they were identified as duplicates, all these SecureDrop sites have very similar characteristics and can thus be expected to be identifiable at a similarly high rates as the POGO site. In particular, we noted that these pages embed images and use scripts and CSS styles that make them large and therefore distinguishable.

It can be argued that the existence of various similar SecureDrop sites creates an anonymity set and makes some sites cover up for each other. On the other hand however, it may be enough for the adversary to ascertain that the user is visiting *a* SecureDrop site for the anonymity of the source to be compromised.

We did a small, manual analysis of some of the most and least fingerprintable sites (by F1 score) to see if there were any strong correlations with content. We found that pages at the bottom end of the spectrum were smaller and simpler (a hidden wiki, a listing of a directory, nginx config page, etc.) whereas the most fingerprintable pages were larger and more complex (a bitcoin faucet site, a forum, the weasyl art gallery site, propublica, a Russian escort service site). Pages in the middle of the spectrum varied, but were often login pages. It is worth pointing out that the onion services ecosystem has a 90's, GeoCities "look," where pages tend to be simple HTML and sites that do not follow this aesthetic will stand out.

[8] `https://securedrop.pogo.org`

8 Limitations and future work

With 482 onion sites, this is the largest website fingerprinting study of onion service sites. Even so, our results may not be representative of the entire onion service universe. We made our best effort to collect as many onion service URLs as possible using `ahmia.fi`. While there are more effective methods to collect .onion addresses, such as setting up a snooping Hidden Service Directory [24], they are ethically questionable.

Our data is a snapshot of the onion services space over 14 days. As the onion services change constantly, and fingerprintability depends not just on individual sites but the whole set, the dataset and the analysis should be updated regularly for a diagnosis of current levels of fingerprintability.

As new website fingerprinting attacks are proposed, features that are important to fingerprintability now may become less so, especially if defenses are introduced or if the design of websites changes. The methods introduced in this paper for extracting features and understanding what makes certain sites identifiable, however, are a lasting and relevant contribution. In particular, we argue that the effectiveness of a proposed defense should be examined not only on average, but that it should account for possible disparate impact on different sites depending on their features. For example, even if a defense significantly lowers the average accuracy of a website fingerprinting attack, it could be that certain sites are always correctly identified, and thus left unprotected by the defense. We also point out that we focus on whether a site blends well with other sites, triggering frequent misclassifications in the context of website fingerprinting attacks, and that the effectiveness of using such techniques as basis for defending against website fingerprinting, has dependencies on the actions taken by other onion services.

Our data collection methodology follows standard experimental practices in the website fingerprinting literature when crawling only home pages. On the one hand, limiting the evaluation to home pages (rather than including all inner pages of a site) reduces the classification space and gives an advantage to the adversary compared to considering that users may directly browse to the inner pages of a site. We argue that a fraction of users will still first land on the homepage of a site before visiting inner pages and thus this adversarial advantage is not unrealistic. We also note that the link structure of inner pages in a website can be exploited to improve the accuracy of website fingerprinting attacks.

Compared to using `wget`, `curl` or headless browsers, our Tor Browser based crawler better impersonates a real browser, limiting the risk of differential treatment by onion services. Still, it is possible detect the presence of Selenium based automation using JavaScript.

The adversary can sanitize training data by taking measures such as removing outliers, but cannot do so for test data. Since we measure an upper bound for the fingerprintability of websites, we sanitize the whole dataset including the test data. Note that this is in line with the methodology employed in prior work [21, 28].

We acknowledge that redesigning a site to be small and dynamic, as suggested best practice by our analysis, may not be an option for some sites for a variety of reasons. This is a limitation of our approach to countermeasures, but might be a limitation to website fingerprinting defenses in general, as large sites are easily identified by website fingerprinting attacks. However, we believe that our results can inform the design of application-layer defenses that alter websites in order to perturb site-level features [8]. This would allow to optimize existing application-layer defenses by focusing on the features that our site-level feature analysis has identified as most identifying, thus reducing the performance that these defenses incur in Tor.

Previous studies on website fingerprinting have shown that data collected from regular sites get stale over time, namely, the accuracy of the attack drops if the classifier is trained on outdated data [15]. For onion services, Kwon et al. did a similar experiment and showed that onion services change at a lower rate than regular sites and do not get stale as quick [17]. For this reason, in this paper, we assume the adversary can keep an updated database of templates.

Reducing the accuracy of website fingerprinting attacks can be framed as an adversarial learning problem. A webpage can be redesigned to modify its site-level features (especially those that contribute the most to fingerprintability) to trick the classifier into making a misclassification. In future work we plan to tackle finding efficient ways to altering these website features to launch *poisoning attacks* against website fingerprinting classifiers [14] under constraints such as bandwidth, latency and availability.

Finally, we acknowledge that the random forest regression method to determine the fingerprintability of a webpage given only web-level features is currently useful only for feature analysis. This is due to a number of factors, such as removing the middle of the spectrum sites and balancing the priors. Although there are a few challenges and limitations, creating an accurate tool that can determine if a site will be easily fingerprinted from only site-level features would be very valuable to onion services.

9 Conclusion

Our work intends to change the way that we build and analyze website fingerprinting attacks and defenses, and differs from previous website fingerprinting contributions in several ways. We do not propose a new attack algorithm (with the exception, perhaps, of the ensemble method) or an explicit defense, but study instead what makes certain sites more or less vulnerable to the attack. We examine which types of features, with intentional generality, are common in sites vulnerable to website fingerprinting attacks.

This type of analysis is valuable for onion service operators and for designers of website fingerprinting defenses. A website fingerprinting countermeasure may have a very disparate impact on different sites, which is not apparent if only average accuracies are taken into consideration. Further, we note that from the perspective of an onion service provider, overall accuracies do not matter, only whether a particular defense will protect their site and their users. Our results can guide the designers and operators of onion services as to how to make their own sites less easily fingerprintable, in particular considering the results of the feature analyses and misclassifications. For example, we show that the larger sites are reliably more identifiable, while the hardest to identify tend to be small and dynamic.

This work is also a contribution to adversarial machine learning. Most work in adversarial learning focuses on attacking a specific algorithm and feature set, but in many privacy problems this model does not fit. Our study investigates methods to force the misclassification of an instance regardless of the learning method.

References

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

[2] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, ASIACCS '06, 2006.

[3] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[4] Xiang Cai, Rishab Nithyanand, and Rob Johnson. CS-BuFLO: A congestion sensitive website fingerprinting defense. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 121–130. ACM, 2014.

[5] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *ACM Conference on Computer and Communications Security (CCS)*, pages 227–238. ACM, 2014.

[6] 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.

[7] 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`.

[8] Giovanni Cherubin, Jamie Hayes, and Marc Juarez. "Website fingerprinting defenses at the application layer". In *Proceedings on Privacy Enhancing Technologies (PoETS)*, pages 168–185. De Gruyter, 2017.

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

[10] 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 *IEEE Symposium on Security and Privacy (S&P)*, pages 332–346. IEEE, 2012.

[11] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *USENIX Security Symposium*, pages 1–17. USENIX Association, 2016.

[12] 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.

[13] Andrew Hintz. Fingerprinting websites using traffic analysis. In *Privacy Enhancing Technologies Symposium (PETS)*, pages 171–178. Springer, 2003.

[14] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58. ACM, 2011.

[15] 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.

[16] 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 (ESORICS)*, pages 27–46. Springer, 2016.

[17] Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas. Circuit fingerprinting attacks: Passive deanonymization of tor hidden services. In *USENIX Security Symposium*, pages 287–302. USENIX Association, 2015.

[18] Sarah Jamie Lewis. OnionScan Report: Freedom Hosting II, A New Map and a New Direction. "`https://mascherari.press/onionscan-report-fhii-a-new-map-and-the-future/`", March 2017. (accessed: May, 2017).

[19] 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.

[20] Xiapu Luo, Peng Zhou, Edmond W. W. Chan, Wenke Lee, Rocky K. C. Chang, and Roberto Perdisci. HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows. In *Network & Distributed System Security Symposium (NDSS)*. IEEE Computer Society, 2011.

[21] Andriy Panchenko, Fabian Lanze, Andreas Zinnen, Martin Henze, Jan Pennekamp, Klaus Wehrle, and Thomas Engel. Website fingerprinting at internet scale. In *Network & Distributed System Security Symposium (NDSS)*, pages 1–15. IEEE Computer Society, 2016.

[22] 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.

[23] Mike Perry. Experimental defense for website traffic fingerprinting. Tor Project Blog. `https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting`, 2011. (accessed: October 10, 2013).

[24] Amirali Sanatinia and Guevara Noubir. HOnions: Towards Detection and Identification of Misbehaving Tor HSdirs. In *Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs)*, 2016.

[25] 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.

[26] Tao Wang. *Website Fingerprinting: Attacks and Defenses*. PhD thesis, University of Waterloo, 2016.

[27] 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.

[28] 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.

[29] 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.

A Appendices

A.1 Site level features

Table 6 shows the site-level features and statistic used to aggregate each site-level features within a site class. We followed the feature extraction step outlined in Section 3 to obtain the site-level features. Here we present a more detailed overview of feature extraction for different site-level feature families.

Table 6: Site-level features and statistics used to aggregate them across download instances. Nominal and binary features such as `Made with Wordpress` are aggregated by taking the most frequent value (i.e. mode) of the instances. Quantitative features such as `Page load time` are aggregated using median, as is is less sensitive to outliers than the statistical mean.

Feature	Median	Mode	Description
`Number of HTTP requests`	●		Number of HTTP requests stored by the browser add-on
`Number of HTTP responses`	●		Number of HTTP responses stored by the browser add-on
`Has advertisement`		●	HTTP request matching EasyList [9]
`Has tracking/analytics`		●	HTTP request matching EasyPrivacy [10]
`HTML source size`	●		Size (in bytes) of the page source
`Page load time`	●		As determined by Selenium
`Made with Django`		●	As determined by `generator` HTML meta tag
`Made with Dokuwiki`		●	As determined by `generator` HTML meta tag
`Made with Drupal`		●	As determined by `generator` HTML meta tag
`Made with Joomla`		●	As determined by `generator` HTML meta tag
`Made with MediaWiki`		●	As determined by `generator` HTML meta tag
`Made with OnionMail`		●	As determined by `generator` HTML meta tag
`Made with phpSQLiteCMS`		●	As determined by `generator` HTML meta tag
`Made with vBulletin`		●	As determined by `generator` HTML meta tag
`Made with WooCommerce`		●	As determined by `generator` HTML meta tag
`Made with Wordpress`		●	As determined by `generator` HTML meta tag
`Made with CMS`		●	True if any of the "Made with..." features above is true
`Number of audio`	●		As determined by the `Content-Type` HTTP response header
`Number of domains`	●		As determined by the `Content-Type` HTTP response header
`Number of redirections`	●		As determined by the presence of `Location` HTTP response header
`Number of empty content`	●		Number of HTTP responses with `Content-Length` equal to zero
`Number of fonts`	●		As determined by the `Content-Type` HTTP response header
`Number of HTML resources`	●		As determined by the `Content-Type` HTTP response header
`Number of images`	●		As determined by the `Content-Type` HTTP response header
`Number of other content`	●		As determined by the `Content-Type` HTTP response header
`Number of scripts`	●		As determined by the `Content-Type` HTTP response header
`Number of stylesheets`	●		As determined by the `Content-Type` HTTP response header
`Number of videos`	●		As determined by the `Content-Type` HTTP response header
`Number of waterfall phases`	●		Approximate number of HTTP waterfall chart phases as determined by switches from request to response or response to request.
`Screenshot size`	●		Size (in bytes) of the screenshot saved by Selenium
`Page weight`	●		Sum of the HTTP response sizes (in bytes)
`Total request size`	●		Sum of the HTTP request sizes (in bytes)

[9] `https://easylist.to/easylist/easylist.txt`
[10] `https://easylist.to/easylist/easyprivacy.txt`

A.2 Confusion Graph for CUMUL

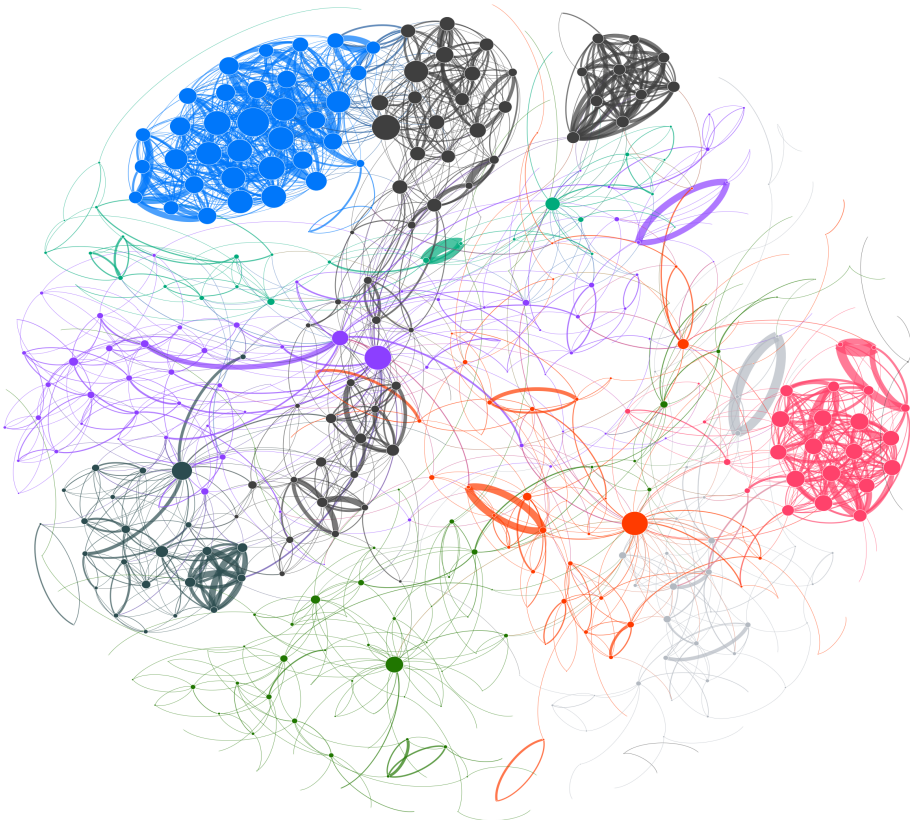


Figure 10: Confusion graph for the CUMUL classifier drawn by Gephi software using the methodology explained in Section 4.6. Nodes are colored based on the community they belong to, which is determined by the Louvain community detection algorithm [3]. Node size is drawn proportional to the node degree, that is, bigger node means lower classification accuracy. We observe highly connected communities on the top left, and the right which suggests clusters of onion services which are commonly confused as each other. Further, we notice several node pairs that are commonly classified as each other, forming ellipses.

Publication

Inside Job: Applying Traffic Analysis to Measure Tor from Within

Publication Data

Jansen, R., Juarez, M., Galvez, R., Elahi, T., and Diaz, C. Inside job: Applying traffic analysis to measure tor from within. In *Network & Distributed System Security Symposium (NDSS)* (2018), Internet Society

Contributions

- Principal author. First and second authors contributed equally.

Inside Job: Applying Traffic Analysis to Measure Tor from Within

Rob Jansen[1], Marc Juarez[2], Rafa Gálvez[2], Tariq Elahi[2], and Claudia Diaz[2]

[1] U.S. Naval Research Laboratory
[2] imec-COSIC KU Leuven

Abstract. In this paper, we explore traffic analysis attacks on Tor that are conducted solely with middle relays rather than with relays from the entry or exit positions. We create a methodology to apply novel Tor circuit and website fingerprinting from middle relays to detect onion service usage; that is, we are able to identify websites with hidden network addresses by their traffic patterns. We also carry out the first privacy-preserving popularity measurement of a single social networking website hosted as an onion service by deploying our novel circuit and website fingerprinting techniques in the wild. Our results show: (i) that the middle position enables wide-scale monitoring and measurement not possible from a comparable resource deployment in other relay positions, (ii) that traffic fingerprinting techniques are as effective from the middle relay position as prior works show from a guard relay, and (iii) that an adversary can use our fingerprinting methodology to discover the popularity of onion services, or as a filter to target specific nodes in the network, such as particular guard relays.

1 Introduction

Tor [8] network entry and exit points have received considerable focus over the years through continuous research activity because the entry and exit points directly connect to the end-user and destination, respectively. However, potential threats from *middle relay* positions have received far less attention. We believe that this is because there is a common misconception that malicious or compromised middle relays are not a significant threat to end-users' privacy since they do not directly connect to either the end-user or the destination and thus are unable to link both parties together. While middle relays are not privy to the network addresses of the client and destination, they can still passively yet directly observe a plethora of other information including service

access times, transfer volumes and data flow directions, and the preceding and succeeding relays chosen for connections. This *information leakage* could be analyzed to discover client usage and network patterns in the Tor network and thus yield potential attack vectors, and we believe that such threats present a wide gamut of possible avenues for research.

In this work we focus on the novel application of different traffic analysis techniques such as circuit fingerprinting [22] and website fingerprinting (WF) [4, 14, 26, 28–30] performed from middle relays as opposed to the usual guard relays. We design the first circuit and website fingerprinting algorithms specifically for use at middle relays, and we are the first to apply machine learning to detect from which of many possible circuit positions a relay is serving. Using our novel *circuit purpose*, *relay position*, and *website fingerprinting* algorithms, we produce a classification pipeline (*i.e.*,, a python library) that can identify which onion service websites accessible through the Tor network are visited by Tor users. In selecting this specific scope we are keen to focus on sensitive usages of Tor (*i.e.*,, onion services) where assumption failures may lead to high-stakes consequences. We are also specific enough to yield a concrete methodology, tangible code, and data artifacts, as well as empirical results that were thoroughly analyzed with well understood limitations. We provide our viable framework of tools[3] so that future work may use them as foundations for the study of attacks and their mitigation.

In the interest of real-world applicability, we deploy our classification pipeline in the first real-world measurement study using WF with real Tor users. For the sake of ethical research and limiting the impact on user privacy, our pipeline is augmented with PrivCount [19], a privacy-preserving statistics collection suite designed to provide differentially-private [9] results, hiding individual user activity. We use PrivCount to measure onion services while focusing on the popularity of only a single well-known social networking platform (SNS) to further limit the impact on user privacy: the site we measure is accessible through a single-hop onion service indicating that the service itself does not require anonymity. Our measurement not only yields useful information about this particular onion service, but it also serves as a proof-of-concept that our classification pipeline can be used at middle relays as a vector for information leakage. We note that while we as ethical researchers are constrained by the differentially-private (*i.e.*,, noisy) results, a malicious actor is not and would be able to produce measurements of higher accuracy. Therefore, our results represent a lower bound on the information leakage potential of WF at middle relays.

We highlight that in this work we treat website fingerprinting techniques as general tools to identify websites. In contrast, previous work on Tor website fingerprinting assumes a malicious guard relay or a monitored node on the

[3] `https://github.com/onionpop`

client-to-guard network path; under this previous model, it is assumed that an adversary already knows or can easily discover a target client's IP address and all that remains in order to mount a *linking attack* is to identify the website that the client visits. In that setting it constitutes an attack to merely perform WF successfully, whereas in this work we use WF to detect revealing information about the Tor network and its usage. One could then leverage that information, *e.g.*,, to perform wide-scale monitoring of onion site usage or to mount an attack linking client to destination (see subsection 9.2).

Our results include: (i) our *circuit purpose* and *relay position* classifiers achieve 92.41%±0.07 and 98.48%±0.01 accuracy respectively; (ii) our WF at the middle classifier achieves more than 60% accuracy for a closed world of 1,000 onion services, which is competitive with classical WF applications; (iii) our one-class classifier for real-world deployment is bounded to 0.6% false positive rate at a cost of decreasing the true positive rate to 40%; and (iv) our real-world deployment shows that the social networking website's onion service accounts for 0.52% of all onion service circuits created. These results are compelling and provide evidence that WF is viable at the middle relay position, that we can effectively target onion service traffic, and that real-world deployments can yield actionable results.

We make the following contributions in this paper:

1. we design the first classifier to detect relay position, and the first classifier to detect circuit purpose from a middle relay position using a novel feature set that utilizes internal Tor protocol meta-data;
2. we are the first to show that traffic fingerprinting techniques are effective from a middle relay position for both closed-world and one-class open-world problems;
3. we produce a classification pipeline that combines circuit purpose, relay position, and WF classifiers for real-world deployment; and
4. we perform the first measurement study that applies traffic fingerprinting to discover Tor onion service popularity, done ethically with privacy-preserving statistics collection methods.

2 Background

2.1 Tor

Clients use Tor by first telescoping a long-lived *circuit* through a series of three volunteer *relays*: the client chooses a persistent *entry guard* relay as its first

relay (using this same guard for three months before rotating to a new one), chooses a new random *middle* relay, and chooses a last *exit* relay that will allow it to connect to the intended Internet service external to the Tor network. The relays forward traffic in both directions through the circuit to facilitate communication between the client and its communicating peer.

Tor's popularity is partly due to the flexibility provided by its design: the external peer need not run Tor or even be aware that the client is connecting through the Tor network. However, clients who connect to external peers must still rely on the existing DNS and SSL/TLS certificate authentication systems, and the external peers themselves are not anonymous. To mitigate these issues, Tor also develops and maintains an *onion service* protocol, a communication mode in which both the user and its peer run Tor and build circuits that are connected together. All communication is internal to the Tor network, and therefore the user and its peer both enjoy anonymity and end-to-end encryption without relying on external insecure name and certificate authentication systems.

2.2 Onion Service protocol

The protocol that Tor clients and onion services use to establish a connection is as follows. In order to advertise themselves and be reachable by clients, onion services maintain a few long-term circuits that last at least one hour. Relays at the end of these long-term circuits play the role of *Introduction Points* (IP). The addresses of IPs along with other onion service information such as their public keys are stored as *descriptors* in a distributed database formed by Tor relays that have been assigned the `HSDir` flag. To create an IP circuit, the onion service must create a regular three-hop circuit and send an `establish_intro` cell to the last relay. The last relay replies with an `intro_established` cell if it agrees to act as the IP on this circuit.

To establish a connection with the onion service, the client first selects a middle relay to serve as a *Rendezvous Point* (RP) and builds a circuit ending at that relay. The client then sends an `establish_rendezvous` cell to the RP which replies with a `redezvous_established` cell. The client must then inform the onion service of this RP using the service's IP. To learn the IP, the client builds a circuit to an HSDir, sends the service's .onion address that was communicated out-of-band, and receives the onion service's descriptor (including the addresses of the service's current IPs). The client then builds a circuit to one of the IPs and sends it an `introduce` cell which contains the RP's address and a one-time secret, all encrypted with the onion service's public key. The IP relays the cell to the onion service which acknowledges the receipt by sending a `introduce_ack` back to the client.

The onion service decrypts the RP address, builds a circuit to it, and sends it a `rendezvous` cell that contains the one-time secret provided by the client for authentication. The RP relays this cell to the client who will verify the one-time secret and acknowledge receipt to the onion service. At this point, a six-hop circuit exists between the client and the onion service and can be used for application communication. Figure 1 depicts the four main types of circuits that have been created (excluding the HSDir circuit): an onion service to IP, a client to IP, a client to RP and an onion service to RP. Note that the circuits created for this process are dedicated to the onion service protocol and cannot be reused for other communications.

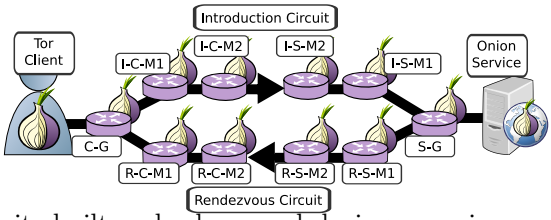


Figure 1: Circuits built and relays used during an onion service connection. The I-S-M2 relay serves as the *introduction point*, and the R-C-M2 relay serves as the *rendezvous point*.

2.3 Stream isolation

Applications tunnel peer connections, called *streams*, through Tor circuits. The Tor software decides if a new stream should be assigned to an existing used circuit, an existing unused circuit, or if a new circuit should be built to handle the stream. Tor would like to provide unlinkability of *unrelated traffic* in order to reduce the exposure to honest-but-curious exit nodes that may track unrelated visits by the same user. However, completely isolating each stream to its own circuit would significantly degrade Tor's performance while allowing malicious servers to cause a client to create an arbitrary number of circuits, which would increase the probability that a client selects a compromised node for at least one of them. For this reason, Tor prefers to isolate groups of streams to their own circuit. In TorBrowser, a hardened fork of Firefox and the recommended web browser to use with Tor, streams are grouped by the first-party domain that appears in the URL bar (across different tabs). This means that almost all streams generated during a page download will go through the same circuit, including requests to third parties. Note that, although unlikely, Tor may create a new circuit while fetching a web page; for example, a restrictive exit policy of a circuit may cause Tor to create a new circuit with an exit that supports the fetch of a particular resource (*e.g.*,, transitions from HTTP to HTTPS and vice versa).

However, the rules for handling onion service traffic are different. Since onion service circuits do not exit the Tor network and require that a *Rendezvous Point* is agreed upon between the client and the onion service, there is currently no stream grouping by the first-party domain of the onion URL in the address bar. Therefore, a user visiting an onion service with mixed first-party onion service and third-party onion or non-onion service content may create multiple circuits to fetch the content; it will create a circuit to fetch all of the first party content, a circuit to fetch all non-onion service third party content (even if each third party is served from a different domain), and a circuit for each third-party embedded resource hosted from a unique onion service.

These peculiarities of the onion service protocol limit the visibility of an adversary monitoring the traffic at the middle: while an adversary at the entry or client-to-entry link is able to capture all the traffic that a user generates during a visit to an onion service, the adversary we consider would only be able to record the traffic for the first-party content.

2.4 Traffic fingerprinting

The traffic analysis techniques that we study in this paper are based on applying supervised learning methods on the encrypted and anonymized traffic traces that are captured from middle relays we control. We study traffic fingerprinting attacks such as circuit and website fingerprinting that use side-channel information leaking from encrypted network packet timing and lengths to discover patterns in the communication. In particular, circuit fingerprinting allows an attacker to distinguish between visits to onion services from regular sites and website fingerprinting enables one to identify the website being accessed. To the best of our knowledge, all previous website fingerprinting studies in Tor have been conducted either at the entry guard or somewhere on the network path between the client and the guard.

Most studies evaluate WF in a *closed world* model in which it is assumed that the classifier could be trained on data for *all* of the sites that the user was possibly going to visit. This assumption is unrealistic because there are potentially billions of websites and the resources necessary to collect, store, and process the data for all such sites would be overwhelming. A more realistic evaluation method uses an *open world* model in which only a small fraction of the sites are available to the adversary for training. However, the closed world *has* been considered a realistic scenario *if* the adversary aims at detecting only onion services [6]. It has been shown that a local and passive adversary can effectively first detect onion service visits using circuit fingerprinting, and then apply website fingerprinting methods to infer to which website they belong [22, 27].

In this paper, we evaluate the effectiveness of traffic fingerprinting from Tor middle relays under both open and closed world models while focusing on onion services. WF is particularly threatening for onion services for two reasons [6, 25]: (i) there are fewer onion services than regular sites and, since the adversary can filter out visits to regular sites, it needs less resources to fingerprint onion services effectively; and (ii) onion services hide their location on the network so that it is difficult to censor them and may host sensitive content, and therefore visitors of those sites are especially vulnerable if the WF attack is successful.

3 Requirements and ethical research

We now describe the capabilities required to fingerprint onion service websites from the middle relay position and discuss ethical considerations.

3.1 Requirements

To apply the techniques described in the following sections, we do not depend on the ability to break the encryption of Tor but do depend on the ability to eavesdrop on all network traffic to and from relays we control. We can obtain a traffic *trace* or *sample* of both the encrypted network packets and the Tor protocol messages (*i.e.*,, *cells*). We are able to observe, decrypt, and read the headers and payloads of Tor cells that are destined for the middle relays we control, but we can only observe and read the headers of cells intended for another destination and forwarded through our relay (the payloads of such cells are encrypted).

We also need to deploy at least one middle relay that contributes bandwidth to Tor. Our attacks become more statistically sound as we observe more circuits, and the fraction of circuits that we will observe roughly correlates with the amount of bandwidth that we contribute. Note that it is quite affordable to run Tor relays in dedicated servers or as virtual instances on the various low-cost cloud platforms available.

Furthermore, we only require *local* visibility of the network because we can only observe circuits from clients that pick our relays and cannot observe other activity. Figure 2 depicts the position of the relay from which we perform the website fingerprinting that is the focus of this paper.

We desire to be able to utilize the machine learning techniques we propose on common desktop hardware. This means that along with the ability to collect data (described in more detail in subsection 6.4), we can also perform the

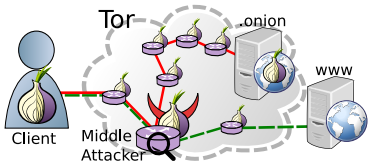


Figure 2: The adversary runs a middle relay and monitors the Tor messages that it relays. The adversary will observe circuits carrying traffic for onion services internal to Tor and regular web servers external to Tor.

pre-processing and data cleansing, the training, and finally the classification tasks all on hardware commonly found on desktop computers of today.

3.2 Ethical Considerations

We have contacted the Tor Research Safety Board[4] for advice on the ethical implications of this research and have followed their recommendations. We also contacted the SNS that we have taken as use case for this study for responsible disclosure but did not receive any response from them. Since we investigate a number of settings with varying levels of risk for real users, we provide additional details about ethical research throughout the paper.

4 Advantages of the middle of the path

In this section, we discuss the benefits of running middle relays to analyze onion service traffic as compared to relays that are at the ingress and egress points of the network.

4.1 Exit

Exit relays make connections outside of the Tor network, and an exit relay will never be chosen by Tor's default path selection algorithm in non-exit positions or in onion service positions due to exit relay bandwidth scarcity. An exit relay will thus not be useful for our purposes, given that it will not route any onion-service visit.

[4] `https://research.torproject.org/safetyboard.html`

4.2 Guard

Each client chooses a relay from the set of all relays with the guard flag and uses it as its first hop entry into the Tor network for all circuits it builds. A guard relay may serve in both the first-hop guard position and in the middle relay positions, and its bandwidth is split among these two positions. In order to be eligible to serve as a guard, a relay is required to be stable and have high up-time relative to other relays. Additionally, a guard relay will not be fully utilized when it first becomes a guard, because clients only drop their current guard and rotate to new ones after two to three months (there exists a proposal to increase this time to nine months [7]). As a result, it will take several months to reach steady state, and during that time the relay will observe less traffic than in other positions. A guard will observe many circuits, including onion service circuits, from a smaller slowly churning set of clients.

4.3 Middle

A middle relay can be used for any circuit, and may potentially observe the traffic of *any* Tor user in the network, given that enough circuits are made over time. This is in contrast to guard relays that can only observe the traffic of users that have picked them as their first hop.

We are particularly interested in regularly visited onion services. The following equation shows that the probability of a particular middle relay observing a client's circuit increases as the client builds more and more circuits over time, where l is the likelihood of picking that middle for a single circuit and c is the number of circuits that the client has made so far.

$$P(observed) = 1 - (1 - l)^c$$

We investigate how the frequency of visits to an onion site, $f = \frac{c}{t}$ where f is the fraction of the number of visits c in a given unit of time t, affects the probability of being observed by a middle relay. Let's assume that a user visits onion services just once every unit of time, for instance once per day. From the line labeled $f = 1$ in the left plot of Figure 3 we see that this client will have an almost 80% chance of making *at least* one circuit through the malicious middle relay after 1,000 days, or two years and nine months. In contrast, a user that visits onion services ten times per day has the same chances in just a little over three months. As point of reference a similarly provisioned guard relay, shown as the rightmost line labeled $f = 10, guard$, reaches similar levels of probability only after two orders of magnitude of time later, 10,000 days.

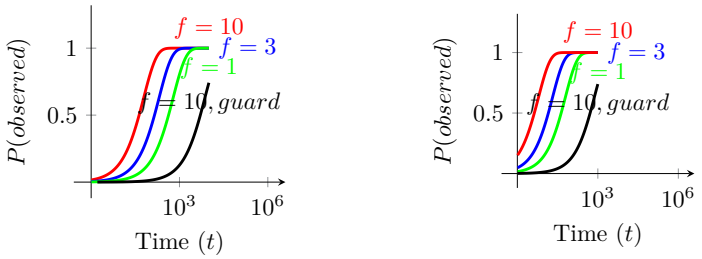


Figure 3: Probability of a client making a circuit using a malicious node with 0.16% (2 MB/s) and 1.6% (20 MB/s), left and right plots respectively, of the middle bandwidth. Plots are shown for various frequencies of visit within a fixed time interval.

This time may be shortened, for instance, by operating ten identical relays—which is not an onerous burden on resources nor difficult to practically achieve—and then there is an 80% chance that a user who connects to an onion service once per day would take about 100 days and a user that connects ten times per day will now only take about 10 days to create at least one circuit through our middle relay (see lines labeled $f = 1$ and $f = 10$ in the right plot of Figure 3). As reference, a similarly provisioned guard in this setting, the rightmost line labeled $f = 10, guard$, has the same probability of observing that client only after two orders of magnitude later, 1,000 days.

The preceding illustrates that middle nodes can enumerate more clients in a shorter time frame. We want to clarify that both the guard relay and middle relay observe the same number of *circuits*, but a different set of *clients*. The guard relay will only observe circuits from a somewhat static subset of Tor clients, but the guard observes all circuits from that subset. In contrast, the middle relay will observe circuits from the set of all clients but only some of the circuits built by those clients (in the same time frame). However, when dealing with a frequently visiting user, the middle relay will be able to obtain a representative sample of these accesses, which provides it qualitatively the same information as the guard. In this way middles have a better overview of the entire onion service and Tor userbase activity, albeit sampled at a known rate.

5 Circuit purpose and position fingerprinting

We have argued that the middle relay position is advantageous for obtaining a statistical sampling of client activities across the Tor network. In this section,

we show how machine learning classification techniques can be used by the middle relay to determine which middle position and which circuit purpose (*i.e.*,, onion service or general) it is serving, enabling it further analyze only circuits that carry onion service traffic.

5.1 Methodology

There are multiple middle onion service circuit positions in which a middle relay could serve, and a middle relay will also serve in non-onion service, general purpose circuits (see Figure 1). To understand how a middle relay can detect its position in a circuit and the circuit purpose, we first generate a large data set of circuits that were built using Tor. We modified a Tor middle relay[5] to log messages containing the information necessary to perform the classification, and we incorporated a new signaling mechanism that enables the client to send ground truth to the relay for evaluation purposes. We run our modified Tor code under simulation in Shadow [18] as well as on the live Tor network. For the latter, we need to be sure that we do not capture any information from circuits that are not under our control in order to protect real Tor users.

Circuit Signaling

In order to perform classification, our middle relay requires timing and flow information from Tor circuits. A middle relay and client under our control will not be directly connected, however, and therefore we need a signaling mechanism with which our client can identify to our middle relay the circuits that we control and that are safe to analyze.

We added a new *signaling* cell to the Tor protocol and a mechanism to allow the client to pin a relay as its R-C-M1 middle on all circuits. The signaling cell is inserted by our Tor client into new circuits that it creates through our middle. The new cell is encrypted for and sent to our middle relay through the Tor circuit, and no other relay in the circuit can read it. The new cell identifies to our middle relay that the circuit on which the cell is sent should be labeled as our own circuit and therefore that it is safe to start tracing the circuit. The signal cell may include an optional payload so that the client can send ground truth information about the circuit (*i.e.*,, the purpose and position of the relay), stream, and HTTP request (*i.e.*,, the URL being fetched) to our middle relay.

[5] We branched Tor at version `0.2.7.6`

Tracing Traffic Patterns at the Middle Relay

Once our middle relay has identified that a circuit is initiated at our client, it begins collecting information about the circuit and the cells transferred through it. This information is exported through the Tor control protocol, which provides a well-defined interface [1] for other applications to request and consume information about Tor (including information associated with periodic events). For every circuit on which we receive a signal from our client, the middle relay exports a unique circuit ID, circuit creation time, as well as the IP address, fingerprint, and relay flags of the previous and next hop relays. It also begins logging information about each cell it sends and receives on that circuit: it logs the direction of the cell (outbound or inbound), how the cell was transferred (sent or received), the time the cell was transferred, the unique ID of the circuit to which the cell belongs, and the cell type and command (which are used to instruct relays to, *e.g.*,, create, extend, and destroy circuits or relay traffic).

Collecting Data with Shadow

We ran our customized Tor software under simulation in Shadow [18] in order to generate a large corpus of circuits suitable for analysis. Shadow is a discrete-event network simulator that directly executes Tor, and therefore faithfully executes all of the logic associated with building Tor circuits. Shadow allows us to construct and run a private Tor network, complete with directory authorities, relays, clients, servers, and onion services, and gives us full control over our network. We run Shadow experiments in a private, local environment free of privacy risks to real users.

Our primary goal is to collect a large sample of circuits from the perspective of a middle relay, which could be done either by running many smaller experiments in parallel or fewer larger experiments sequentially using the same amount of RAM. We didn't believe it was necessary to run a full-scale Tor network because the features used by our purpose and position classifiers are not sensitive to network congestion or scale (we don't use any time-based features). Running smaller networks means we can more effectively parallelize our experiments, sample more initial random seeds, and more quickly obtain results. Therefore, we generated a small private Tor network configuration with 50 relays, 128 web clients, 42 bulk clients, and 100 servers. We ran 83 experiments with distinct seeds for one simulated hour each (83 simulated hours in total) on a machine with a total of 40 Intel Xeon 3.20GHz CPU cores (80 hyper-threads) running the latest CentOS 7 version of Linux. We ran 4 multi-threaded experiments at a time, and each experiment took roughly 3.5 hours to complete.

During our experiments, the clients behave as follows. The bulk clients continuously download 5 MiB files. The web clients request data according to an HTTP model where the size of the first request and response, the number of embedded objects per page, the size per embedded object request and response, and the number of distinct domains per page are all sampled from the HTTP Archive.[6] Web clients pause for a time drawn uniformly from 1 to 30 seconds after each full web page has been downloaded before starting the download of another page. Because HTTP Archive data is constructed by downloading real websites from the Alexa top sites list, we believe that the number of Tor circuits that are created when clients use this model is representative of typical Tor circuit usage. Finally, 8 of the web clients and 2 of the bulk clients download their data from onion services, while the remainder of each download their data over regular 3-relay-hop circuits.

We configured one middle relay to act as a middle measurement relay. We enabled the signaling mechanism described above on each client in the network, so that our measurement middle relay would receive ground truth circuit information and collect cell traces for all circuits built through it. A significant advantage of using Shadow is that we are able to inspect all such circuits without risking user privacy. During our experiments our middle measurement relay collected tracing information for 1,855,617 total circuits, 813,113 of which were onion service circuits.

5.2 Feature extraction

We extracted features from our large corpus of circuits based on the observation that cell meta-data, such as *cell type* and *relay command*, leaks information to a relay about its position in the circuit and the circuit type (see Figure 4). We also make the following observations: (i) a relay will send a different number of cells during the circuit construction process depending on its position in a circuit; (ii) different relay positions may receive different cell types and relay commands during circuit construction (*e.g.*,, guards and middles will *extend* circuits while exits will not); (iii) relays may or may not connect to other known relays on either side of a circuit (iv) onion service introduction circuits transfer much less data than rendezvous circuits used to download web content; and (v) asymmetric web content downloads would result in more cells traveling toward the circuit originator on client-side rendezvous circuits but away from the circuit originator on service-side rendezvous circuits. To incorporate the previous observations, we use as features the counts of the number of each possible (cell type, relay command) pair that a relay handles. A relay may

[6] `http://httparchive.org`

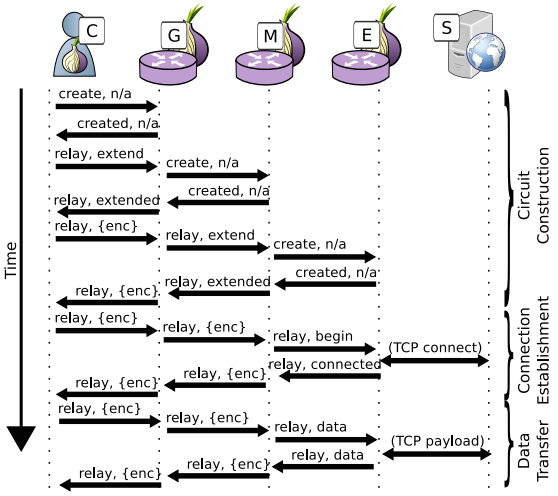


Figure 4: Circuit establishment and data transfer involves various cell *types* and *commands*, which in this example are labeled on the arrows and are readable by the node to which the arrow points ({enc} denotes that the command is unreadable). Cell meta-data leaks information to a relay about its position in the circuit and the circuit type.

Figure 5: A relay may send and receive a different number of cell types on the inside and outside of a circuit.

observe a cell on different sides of a circuit (see Figure 5), so we also include counts of the number of cells that a middle is sending and receiving on both the inbound initiator side of the circuit (inside) and the outbound extension side of the circuit (outside). Finally, we include the total number of cells that the relay is sending and receiving on either side of the circuit, and whether or not the previous and next hops may serve as guard or exit relays.

Note that although it was recommended in previous research [22], we do not include cell timing information in order to be more robust to Tor relay congestion. Interestingly, during the feature analysis we incorporated the circuit duration and cell sequence features that Kwon *et al.* have shown to be useful in distinguishing circuit purpose when classifying circuits from the position of the guard-to-client link [22]. However, we found that these features *reduced* the accuracy of our classifier, and therefore we ignore those features in the remainder of our analysis.

5.3 Training

The goal of our classifier is to predict when a relay is serving on a rendezvous *purpose* circuit and in the first middle relay *position* (*i.e.*,, an R-C-M1 relay as shown in Figure 1). Previous work by Kwon *et al.* [22] provides a decision-tree driven classifier to perform *purpose* filtering but it was designed for use by relays in the guard position or an eavesdropper on the client-to-guard path and not from middle relays. We design a new random-forest driven classifier that performs with comparable accuracy but is tuned for relays in positions between the guards of the client and onion service. Random forests generalize better than simple decision trees, which tend to overfit the training data, thus random forests are more robust against small differences between training and testing settings. We followed prior work on model selection and tuning [14, 22] and, after a search of the parameter space, we found that 30 trees for the random forest provide the highest accuracy.

We trained separate random forest classifiers for circuit purpose and circuit position using the `pyborist` and `sklearn` python APIs on our Shadow-generated circuit dataset and the features we previously described. For both classifiers, we assume no prior knowledge about the circuit purpose or position, so the classifiers could be run independently of one another without affecting the accuracy. To ensure that the classifier does not overfit to our specific dataset, we used standard machine learning procedures such as balancing the dataset so that each class (*i.e.*,, rendezvous vs. other purpose, and C-M1 vs. other position) has the same number of circuits.

We used k-fold cross validation ($k = 10$) to measure how well the classifiers generalize to unseen data. During this process, our original circuit sample is randomly partitioned into k equally-sized subsamples. There are k phases in total: in each phase, a distinct subsample is used as the *testing set* while the remaining $k - 1$ subsamples are used as the *training set*. To train, we convert each circuit from the training set into a feature set labeled with the true class (the true purpose or position) and pass that into the classifier's training function. To test, we convert each circuit from the testing set into a feature set (without the ground truth class label) and pass it into the classifier's prediction function to predict the class label. We evaluate prediction performance by measuring true and false positives and negatives and computing standard related metrics such as accuracy and precision.

5.4 Results

The evaluation results are shown in Table 1. As shown, the accuracy for the purpose classifier is over 92 percent with a standard deviation of 0.07 and the accuracy of the position classifier is over 98 percent with a standard deviation of 0.01. Table 2 shows the most important features as determined by our analysis, *i.e.*,, the features that minimize the information gain in every branch of the random forest. Not surprisingly, cells associated with the circuit construction (create/created type cells and relay type cells with extend/extended commands) are often some of the top features for distinguishing both purpose and position, and the total number of cells sent and received are also useful for both classifiers.

Table 1: 10-fold cross-validated circuit classification results.

	Purpose (rendezvous vs other)	Position (C-M1 vs other)
Accuracy	$92.41 \pm 0.07\%$	$98.48 \pm 0.01\%$
Precision	$91.87 \pm 0.11\%$	$97.16 \pm 0.03\%$
Recall	$93.05 \pm 0.09\%$	$99.88 \pm 0.01\%$
F-1	$92.46 \pm 0.07\%$	$98.50 \pm 0.01\%$
True Positives	396,615 (91.77%)	821,478 (97.08%)
False Positives	35,576 (8.23%)	24,689 (2.92%)
False Negatives	30,056 (6.95%)	984 (0.12%)
True Negatives	402,135 (96.05%)	845,183 (99.88%)

Based on our results, we believe that our simple cell-based counters serve as effective features for position and purpose classification. They are simple and easy to compute and may potentially be useful in other contexts such as onion service fingerprinting when access to Tor cell meta-data is available.

6 Onion Service fingerprinting

In this section, we explore the extent to which middle relays can be effective at carrying out state-of-the art website fingerprinting techniques on onion sites. We describe how we modify and use the Tor and `tor-browser-crawler` software to

Table 2: Most important circuit classification features*.

Purpose (rendezvous vs other)	Position (C-M1 vs other)
13.73% # (relay,{enc}) cells	23.22% next node is relay
11.11% # (create2,n/a) cells	11.23% # (relay,extended2) cells
09.10% # cells sent total	09.26% # (created2,n/a) cells
08.89% # cells received total	06.66% # cells sent total
08.31% # cells sent inside	06.61% # (create2,n/a) cells
07.78% next node is exit	06.12% # (relay_early,extended2) cells
07.66% # (relay_early,extended2) cells	05.75% # cells received outside
06.78% # cells received inside	05.32% # cells received total
05.83% # (relay_early,{enc}) cells	05.25% previous node is guard
04.26% # (created2,{enc}) cells	04.06% # (relay,{enc}) cells

* Shown are the top 10 of 22 total features used (both classifiers used the same features).

gather data from a middle relay position, explain the fingerprinting techniques that we performed on these data, and how we evaluated their efficacy.

6.1 Evaluating website fingerprinting

The website fingerprinting techniques that we chose for this evaluation are the most scalable and successful known so far in the literature. These are:

Wang-kNN [29]

presented by Wang *et al.*, it achieves over 90% accuracy for 100 non-onion sites. Wang *et al.*'s features were actually families of features defined by a certain parameter—for instance, the number of outgoing packets in the first N (parameter) packets. By varying these parameters they generated more than 3,000 features. The underlying learning model they used was a k-Nearest Neighbors classifier (k-NN). k-NN classifies an instance by averaging over the k closest instances in the dataset according to a given distance metric. In their

case, they used a weighted euclidean distance with a tuning mechanism that minimizes the distance among traffic samples that belong to the same site, a property that is especially suited for k-NN.

CUMUL [26]

presented by Panchenko *et al.*, it is based on an SVM with a Radial Basis Function (RBF) as a kernel. Their evaluations show that CUMUL achieves 93% accuracy for 100 non-onion sites. CUMUL's main feature is the cumulative sum of packet lengths. The cumulative sum of a traffic trace is represented as a vector with as many components as the number of packets in the trace. Recursively, the first coordinate value is the length of the first packet and the i-th coordinate is calculated by adding the length of packet i and the value of coordinate $i-1$. Since SVM expects fixed-size feature vectors and the cumulative sums have varying sizes, they interpolate 100 points for each cumulative sum.

k-Fingerprinting (k-FP) [14]

presented by Hayes and Danezis, it is the most recent website fingerprinting technique. It is based on Random Forests (RF) and k-NN and achieves similar accuracy to CUMUL. Their feature sets are formed by 175 features that, among others, include most of the features that have already been proposed in the website fingerprinting literature to date. Their feature representation is novel: instead of plugging the features directly into a classifier, they instead use the leaves in a trained RF as the representation for classifying with a k-NN with Hamming distance.

All of these attacks have also been evaluated in an *open world* of websites where they perform with high accuracy. The open world is a more realistic setting where the adversary cannot train on all sites that can be visited by the victim.

6.2 Methodology

We gather data that enables us to analyze the effectiveness of onion service website fingerprinting attacks from internal circuit positions. We do this by running our modified Tor software described in section 5.1 and section 5.1, crawling a set of known onion sites, and tracing our client's circuits from our own middle relay.

We have automated our crawls using a web crawler that visits a list of onion service URLs with the Tor Browser, called `tor-browser-crawler`.[7] We based our collection methodology on previous studies on website fingerprinting. As Wang and Goldberg proposed [30], we divided the crawls into batches. In each batch, we iterate over the whole list of URLs, visiting each URL several times. The rationale behind batched crawls is that visits to a page in different batches allows the capture of features that are invariant over time; and combining visits within a batch reduces the time-independent noise due to sporadic errors or per-visit variations such as advertisements. We also disable the `UseEntryGuards` Tor option so that we select a different entry guard for each circuit. As a result, we significantly reduce the probability that our testing and training instances are collected over a circuit with the same entry guard, which would unrealistically improve the accuracy of the attack [20].

To speed up the total crawling time, for every visit we create a new identity using Tor Browser's `torbutton` add-on, and then signal the Tor controller to select a random entry relay. This way we don't restart Tor on every visit. In addition, restarting the identity guarantees that we have a clean browser state for the visit, as previous studies have pointed out that keeping the browser state may create artificial dependencies between consecutive visits [30].

The client logs TCP packet headers with `tshark` during each visit to an onion page. We ignore the TCP payloads because they are encrypted and thus not useful. By sniffing network traffic, we can reproduce previous WF evaluation techniques that do not allow access to cell-level information. Because we are interested in cell-level information in this work, we also use OnionPerf[8] to collect Tor cell traces at the client. For debugging and error detection purposes, we take a screenshot of the page as rendered by the Tor Browser, intercept and dump HTTP requests and responses with a browser add-on that the crawler install on Tor, and dump the `index.html` source code. Recall that we apply these techniques only on our own circuits and not those of regular users.

As we described in section 5.1, the client pins one of our middles as the R-C-M1 relay (see Figure 1). Our middle relay collects the information from our custom signaling cells as described in section 5.1 using OnionPerf in `monitor` mode. OnionPerf will produce a log file containing the data sent by the client as well as other standard Tor events (*e.g.*,, bandwidth information). Each circuit that is created by our own crawler will be labeled as such in the OnionPerf log file. We later process these raw log files as necessary to apply the website fingerprinting technique.

In addition, our crawler also flags the start of a visit and sends a unique visit ID to the middle along with the ID of the circuit used to carry the first HTTP

[7] `https://github.com/onionpop/tor-browser-crawler`
[8] `https://github.com/robgjansen/onionperf`

request. When we parse the logs, we discard all other circuits built to fetch that onion site. We need these IDs so that we can parse only the cells that go through that first circuit and discard cells to other circuits. (Recall from subsection 2.3 that our middle relay would miss third party onion service circuits, whereas the entry relay will be able to record all clients' circuits.) In fact, due to Tor's stream isolation, the middle relay has the advantage that traffic to the first-party onion service will not blend with traffic to other sites, eliminating the need to use special parsing techniques [31]. Note that our custom signaling cells will be present in the client `tshark` logs, however, we filter out these artificially added cells before classifier training.

The list of URLs that we crawl has been obtained from the ahmia[9] Tor onion service search engine maintainers. Before starting the crawls, we used `torsocks` and `curl` to remove from the list onion sites that were down. We have removed all the screenshots after error detection to avoid keeping illegal data on our hard drives. In total we ran four middle relays to crawl 5,000 different onion websites in parallel. After removing failed visits and thresholding so that all websites had the same number of instances, the dataset ended up having 2,500 onion sites and 80 instances per site.

6.3 Ethics

Safety

We have tested our Tor source code modifications using Shadow [18]. However, Shadow does not run the Tor browser crawler that we require to crawl onion services in the website fingerprinting experiments. In order to capture the complexities of the Tor Browser, and also to evaluate our attacks under realistic background traffic and network congestion conditions, we conduct our experiments in the live Tor network.

The signaling mechanism described in section 5.1 ensures that we only collect traffic generated by our crawler. Using OnionPerf, we log only the events associated with the traffic generated by our client. Thus, analysis and potential attacks will be applied only on traffic data generated by our own visits.

Following the principle of data minimization, our middle relays only collect traffic data attributes strictly necessary for applying traffic fingerprinting attacks. We ignore the payload of network packets captured at our client, as they are encrypted and are not useful for fingerprinting purposes. The HTML sources and screenshots are also removed after the error detection and outlier removal phases.

[9] `https://ahmia.fi/`

Benefits and Risks

Since we are not collecting any data of regular Tor users, there is no de-anonymization risk from our traffic datasets. There may be a small indirect risk of leaking user personal data in the screenshots and HTML sources, but they were deleted before publication, after being used for the integrity checks of our traffic dataset.

With respect to the impact of our experiments on Tor's performance, the volume of the traffic generated by our crawls is comparable to that from a regular user actively browsing the Web for a few hours. We do not expect a significant impact on network performance.

Our methodology allows us to explore one of the main research questions in this work: whether fingerprinting is effective in the middle position of our circuits. In addition, it will help us compare the effectiveness of these techniques at different layers of the network stack (*i.e.*,, the application layer and the transport layer). Previous studies only applied WF on TCP packets and used heuristics to filter cell types that are not useful for fingerprinting (*e.g.*,, `SENDME` cells). Our middle relays have access to cell information and thus can directly utilize or filter control cells that are not related to a website.

6.4 Results

Website Fingerprinting Effectiveness at the Middle

Here we explore the following research question: *how effective is website fingerprinting at the middle with respect to the client*? Specifically, we design an experiment to determine whether the accuracy of onion service fingerprinting is affected by the position in the circuit (*i.e.*,, middle relay as compared to the entry link).

We follow the methodology outlined in the previous section to obtain two datasets: (i) TCP traces as collected between the client and the entry guard and (ii) cell traces as collected from the middle relay. Both sets of traces were collected at the same time to avoid confounding variables like changes of the website over time [20]. To evaluate the effectiveness of website fingerprinting at the middle, we apply the state-of-the-art techniques on both datasets and compare the success rates.

Table 3 shows the accuracy scores for three classifiers on the network traffic data collected at the client. The accuracy is defined as the number of True

Table 3: 10-fold cross-validated accuracies for the three state-of-the-art attacks on our *client-side* TCP traces. The evaluations are closed worlds of 10, 50 and 100 onion sites.

Num sites	k-NN (%)	k-FP (%)	CUMUL (%)
10	$95\% \pm 0.03$	$95\% \pm 0.06$	$92\% \pm 0.04$
50	$75\% \pm 0.02$	$85\% \pm 0.03$	$81\% \pm 0.02$
100	$67\% \pm 0.01$	$68\% \pm 0.03$	$64\% \pm 0.02$

Positives—test instances that have been correctly classified—over the total, also known as True Positive Rate (TPR) or Recall.

As we see in the table, k-FP outperforms the other techniques by a small margin followed by CUMUL and, lastly, k-NN, the least accurate technique. These accuracies are consistent with existing evaluations of these techniques on onion service sites [6]. We also evaluated the techniques on existing datasets [29] to make sure that we are able to reproduce previous evaluations on regular sites and that we do not introduce errors that stem from our methodology; we did not find any major discrepancies from previous results.

On the other hand, Table 4 shows the accuracies the techniques achieved when applied on the cell traces collected from our middle relay. In this table we see that the classifiers are ranked in the same order as in the client: k-FP being the most accurate and k-NN the least accurate. With respect to the accuracies obtained at the client for the same classifier, we see some interesting differences: while k-NN has a few percent points decrease in accuracy with respect to the entry link scenario, both k-FP and CUMUL perform a few percent points better when they are applied at the middle. This is plausible because each classifier uses different features that may be more or less robust to timing and order differences between both positions. The accuracy improvement can be explained by the fact that we used TCP traces at the client, whereas the middle dataset includes cell traces, conveying a higher amount of information and including less noise than TCP traces [30].

Another interesting observation is that discarding all third-party circuits for training and testing does not impact classifier accuracy. We attribute this result to the low prevalence of third party embedded content in onion services (it has been found on a large dataset of onion services to be less than 20% overall [6]).

Table 4: 10-fold cross-validated accuracies for the three state-of-the-art attacks on our dataset of cell traces collected at the *middle relay*.

Num sites	k-NN (%)	k-FP (%)	CUMUL (%)
10	$91\% \pm 0.03$	$100\% \pm 0.00$	$99\% \pm 0.03$
50	$73\% \pm 0.01$	$91\% \pm 0.01$	$86\% \pm 0.03$
100	$68\% \pm 0.01$	$76\% \pm 0.02$	$76\% \pm 0.02$
500	$64\% \pm 0.00$	$72\% \pm 0.01$	$66\% \pm 0.01$
1,000	$59\% \pm 0.00$	$56\% \pm 0.01^*$	$63\% \pm 0.01$

* Due to RAM constraints we were not able to evaluate k-FP using the optimal parameters for 1,000 sites which reduced classifier accuracy.

Open world scenario

The preceding analysis and results are applicable to an idealized closed-world scenario where we try to identify known and trained-on onion websites. We now consider a more realistic and challenging open-world setting—one where unseen and unknown onion websites may be introduced at testing time. We now present an enhancement of our website fingerprinting techniques for this scenario.

In other recent open-world evaluations, the classifier is only trained on a small fraction of the web pages in the world. In this setting, the user may visit any page in the world, including pages of which the classifier is not aware. These works define the open world evaluation as a binary problem: the positive class is formed by a set of *monitored* pages that the classifier aims to identify and the negative class by the remaining *non-monitored* pages [4,15,20,21,26,29]. During training, the classifier is shown examples of both monitored and non-monitored pages; however, the majority of the pages in the non-monitored set are not present in the training set.

In this paper we have approached the open world differently. There is no strong support to believe that the non-monitored samples used for training necessarily represent the whole world of non-monitored pages because the sample that is taken to train the classifier is small compared to the population, *i.e.*,, all pages that could possibly be visited. This sample may bias the classifier toward a specific choice of non-monitored pages selected for training or not actually help the classifier discriminate monitored from non-monitored sites. Instead,

we propose to model the open world as a *one-class* classification problem: the classifier only takes instances for the monitored class and draws a boundary between the monitored pages and *all other* pages.

In particular, we have taken the monitored set to be composed by one single web page—the best-case scenario for the adversary in such an open world. We have collected 5,000 instances of a popular social network service (SNS) that is deployed as an onion site and use 3,750 of the samples to draw the decision boundary and 1,250 for testing. We have used 200,000 instances of 2,500 different sites (80 instances per site) for testing the one-class classifier for onion service pages that are not the SNS.

For the one-class classifier we have used `sklearn`'s implementation of one-class SVM with a radial basis function. The one-class SVM is parameterized on ν which defines an upper bound on the fraction of training errors; ν can be used to adjust the trade-off between False Positives and True Positives. We plotted the ROC curve to find a value of ν that maximizes the number of True Positives while keeping a low False Positive Rate. In Figure 6 we can see that $\nu = 0.2$ achieves such a compromise, providing a FPR lower than 1% while the TPR is slightly higher than 40%.

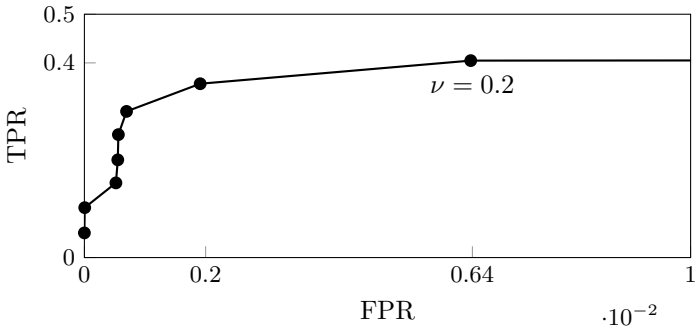


Figure 6: ROC curve to optimize the ν parameter. We can see that $\nu = 0.2$ makes a reasonable trade-off between TPR and FPR. To deal with extreme base rates, it is possible to minimize the FPR at the expense of the TPR.

We have chosen a subset of CUMUL's features because they are also used in an SVM for the closed world problem [26]. After analyzing different subsets, we found that a combination of the first and last interpolation points of the cumulative sum can separate SNS instances from the rest. In particular, we used the second interpolation point (CUMUL's 5th feature), describing the first region of packets in the original trace, and the 87th one (CUMUL's 90th feature), which described mid- and end-regions of the trace.

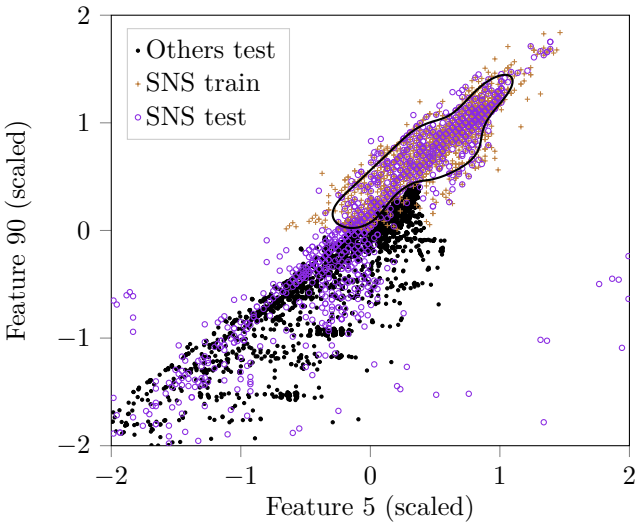


Figure 7: Projection over two CUMUL features of the one-class classification instances. The plus sign marks are instances used for training the classifier, the circle marks are SNS instances used to test the positive class and the cross marks are instances that belong to non-SNS sites used to test the negative class. The black line shows the boundary that was learned by the classifier that minimizes False Positives.

The results are presented in Figure 7 which shows a projection of the classification space on these two features. The orange plus marks are the SNS's training instances, the purple empty circles are the SNS's testing instances, and black filled dots are "Others" instances. The decision boundary learned by the classifier is depicted by the black line.

As we observe in Figure 7, there are many testing samples of SNS that fall outside the boundary. This is because we tuned the classifier to minimize the number of False Positives—instances of non-SNS pages that are classified as SNS. This way we achieve a False Positive rate below 1%, but this has a cost of a large number of False Negatives: the True Positive rate is 40%. The reason we have optimized for low FPR instead of TPR becomes is because we are interested in realistic deployments where the base rate of the positive class becomes relevant, as we discuss next in subsection 6.5.

6.5 Precision is in the detail

The base rate is the probability that a site is visited, and can also be interpreted as site popularity. Previous work has discussed the importance of the effect of the base rate of the positive class (*i.e.*,, the SNS) on the Precision of the classifier [20, 21]. Precision is proportional to the number of samples that our classifier detected as SNS that are actually SNS. In other words, Precision is an estimate of the probability that the classifier was correct when it guessed SNS.

Prior work has pointed out that if the base rate of the positive class (*i.e.*,, the SNS) is orders of magnitude lower than the negative class (*i.e.*,, Others), the False Positive Rate (FPR) has to be negligible so that the classifier can perform with sufficient Precision [20]. Since we cannot estimate the base rate of the SNS's onion site directly for ethical and privacy reasons, we have evaluated the Precision of our one-class classifier for several hypothetical base rates.

In Figure 8, we show the Precision, the TPR, the FPR and training error (*i.e.*,, ν). In the graph we see that all of the metrics are fixed for the whole range of considered base rates, while precision decreases exponentially when the base rate of the SNS tends to zero. The vertical dashed line indicates the base rate (1%) where Precision is 50%; the point where the classifier is correct only half of the time. These results are comparable to previous work that evaluated the precision of the CUMUL classifier and achieved similar results [26].

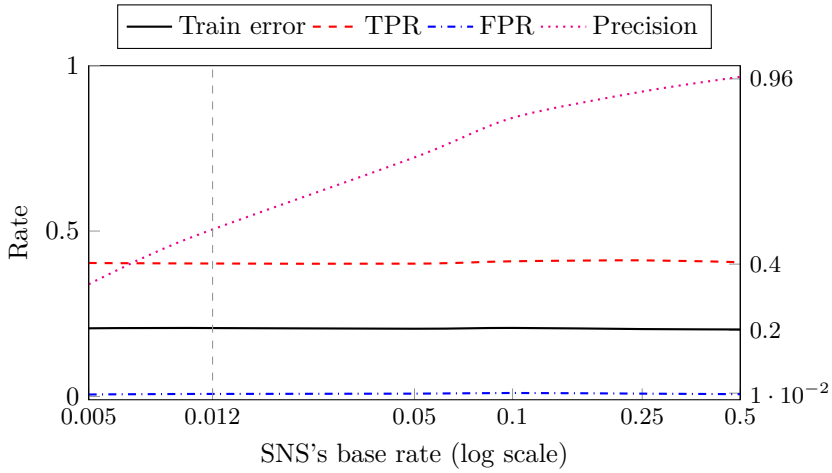


Figure 8: Performance metrics for one-class open world for the SNS's base rates ranging from 0.5% to 50%. The vertical dashed line shows the point in which Precision is 50%.

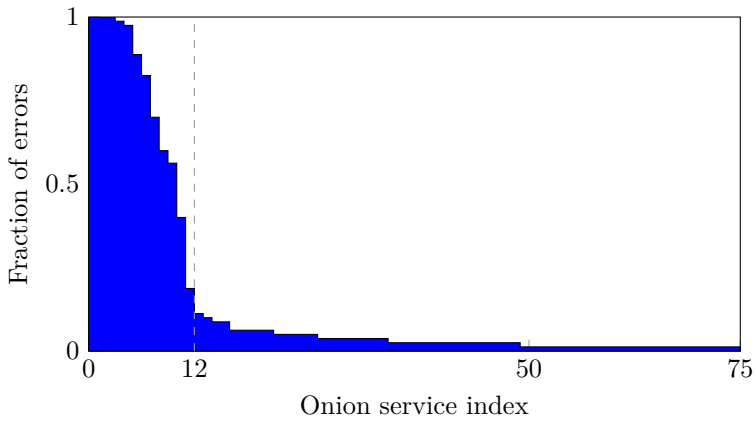


Figure 9: Sites that were confused with the SNS at least once during the classification (2,443 sites had zero errors). Note that the distribution is heavily skewed and 80% of all the errors are concentrated in the 0.5% (12) of the sites (see vertical dashed line) and 3% of the sites (75) include all the errors.

We further analyzed the sites that the classifier misclassified most often. The distribution of errors over the sites is shown in Figure 9. We observe that 80% of the errors are concentrated over 12 of the sites and only 3% of the total number of sites are responsible for 100% the misclassification. Based on this observation, we argue that it is possible that even for 1% FPR (see Figure 8), the classifier may have greater precision if those 12 sites that are responsible for most of the errors are less popular. Note that we assumed a uniform distribution of the sites that belong to the Others sites. Further, it may be possible to design dedicated classifiers that learn to distinguish between the SNS and each of these 12 sites individually in order to reduce the number of False Positives that the classifier incurs overall.

We manually checked the 12 sites that were misclassified as the SNS some weeks after we crawled them. Five of the sites are offline and one of them has been seized by the German Federal Criminal Police. The remaining sites are up and are of a diverse nature: one is a personal homepage, two are movie streaming sites, another is a porn site, one is a hacking page and, surprisingly, the last one is the download page for the SecureDrop whistle-blowing software (`secrdrop5wyphb5x.onion`) run by the Freedom of the Press. We believe that they were confused with the SNS's onion service page due to similarities in page size.

7 Onion Service popularity measurement

We showed in section 5 how a relay can predict that it is serving as a middle (in the R-C-M1 position) and on a rendezvous onion service circuit with high confidence, and we showed in section 6 how website fingerprinting techniques can be used to accurately predict which onion service webpage is visited. In this section, we validate our previous results and show the practicality of the techniques that we developed through a privacy-preserving measurement of the popularity of a social networking site (SNS) accessible as an onion service.

7.1 Measurement goals and methodology

Tor circuit and website fingerprinting techniques have thus far been discussed in the literature in the context of client deanonymization attacks. The goal of the measurement study in this section is to show how to use the classification techniques presented in the previous sections not for client deanonymization, but to predict accesses to and safely measure the popularity of an onion service SNS. In this study, we seek to: (i) develop a reusable framework for safe onion service popularity prediction and measurement; (ii) validate our classification techniques from the previous sections by running them in real time in a realistic public Tor network environment on live Tor traffic (something that has never been done before to the best of our knowledge); and (iii) show how our proof-of-concept measurement framework can be used to discover the popularity of an onion service in the open-world. Note that doing this measurement safely is a primary goal that is further discussed below in subsection 7.4.

Achieving these goals involves several components. First, we run middle relays in the Tor network that provide resources to Tor users. Second, our relays must predict which circuits in which they are serving are onion service circuits (specifically, rendezvous circuits since those are used to access web content). Third, our relays must predict when they are in a middle position of circuits in which they are serving (specifically, the R-C-M1 position since our classifiers were trained for that position). Finally, our relays must predict which of the predicted rendezvous circuits in which they predict the R-C-M1 position are used to access the SNS. We next explain the tools that we built, modified, and used to realize these goals.

7.2 Measurement tools

We enhance PrivCount [19], an existing privacy-preserving Tor measurement tool,[10] to use a new prediction library that we developed to allow us to make predictions in real time on real Tor relay traffic.

PrivCount Overview

PrivCount is a distributed measurement tool, based on the secret-sharing variant of PrivEx [11], that can be used to safely measure Tor. PrivCount works by extracting events from a Tor process and then counting the occurrence of those events across a set of relays. PrivCount consists of a tally server, several share keepers, and several data collectors (one for each relay in a set of measurement relays). The tally server acts as a centralized, but *untrusted*, entity that is primarily used as a proxy to facilitate data transfer between the other nodes and as a single aggregation point following a completed measurement phase. Each data collector initializes each counter that it is configured to count with *differentially-private noise*, and also with *random blinding values* that are secret shared to each share keeper. After initialization, each counter on each data collector and share keeper will appear to hold a random value. The data collectors each connect to their configured Tor relay, extract events, and increment the configured counters when appropriate. At the end of a measurement phase, the data collectors and share keepers send their counter values to the tally server for aggregation. After aggregation, the blinding values that were stored on the share keepers during the measurement will cancel out with the blinding values that were added at the data collectors, and the final output will be the sum of the true counter value and the differentially-private noise that was added by the data collectors.

The noise that is added to each counter protects users under differential privacy [9], and the blinding values provide for secure aggregation *across* measurement relays. No data collector can learn anything about the counters of other relays not also controlled by the same operator, and individual data collector contributions to the final aggregated counter values are hidden (by the random blinding values) as long as at least one share keeper is honest. Jansen and Johnson provide additional PrivCount details and proofs of PrivCount's security and privacy properties [19].

[10] `https://github.com/privcount`

Enhancing PrivCount

PrivCount supports a wide range of statistics (*e.g.*,, number of circuits, amount of data transferred, etc.), and we enhanced it to support counting the real-time predictions of a circuit's purpose, a relay's position in a circuit, and the onion page being accessed in a circuit. To enable these predictions, we utilize a version of Tor that has been modified to allow for circuit signaling as described in section 5.1, and to export the same circuit and cell meta-data that we discussed in section 5.1. We use circuit signaling to collect ground truth during the measurement while separating ground truth circuits that we created from regular circuits that we did not (see subsection 7.5 for more details).

We developed a new library for PrivCount called `onionpop`[11] that implements the classifiers needed for prediction. The onionpop library extracts the features we need for each of the three classifiers from Tor circuits and cells, and wraps the python `sklearn` and `pyarborist` APIs to train the classifiers and predict purpose, position, and webpage. We train our models to make binary predictions of when the purpose is rendezvous, position is R-C-M1, and webpage is the front page of our SNS of interest. We added new counters to PrivCount to record the results of the predictions.

Due to its sensitive nature, we do not log any information to disk from circuits that we did not originate ourselves. Therefore, a significant concern during the development of our prediction library is that PrivCount will need to process cell information from fast Tor relays in real time. For safety, we store circuit and cell meta-data in RAM only for the lifetime of the circuit; when the circuit ends, we run our predictions, increment counters to count the results, and then clear the corresponding circuit and cell meta-data from RAM. This is consistent with PrivCount's data storage model, however, it requires that we process and store a potentially large number of cells. To mitigate potential memory and computational resource bottlenecks, we implement a configurable hard upper limit on the number of cells that we store per circuit (well above the amount we need to distinguish the SNS) and only process a subset of the circuits on our relays by sampling circuits uniformly at random according to a configurable sample rate.

7.3 PrivCount deployment

We set up a PrivCount deployment with 1 tally server, 3 share keepers, and 17 data collectors each connecting to a distinct Tor relay. These nodes were distributed among 3 operators and hosted in 3 countries (Canada, France, and the United States). Each of the relays ran our modified version of Tor, and each

[11] `https://github.com/onionpop/onionpop`

of the tally server, share keepers, and data collectors ran our modified version of PrivCount.[12]

Table 5: Daily action bounds for PrivCount deployment.

Action	Bound
New general-purpose circuits	90
New rendezvous circuits	48
Client-side rendezvous circuits	24
Server-side rendezvous circuits	24
Client-side rendezvous circuits to SNS	2

Privacy

Our PrivCount deployment uses the parameters and privacy budget allocation techniques set out by Jansen and Johnson [19]. Specifically, we use differential privacy parameters $\epsilon = 0.3$ (which has also been used by Tor [13]), and $\delta = 10^{-3}$ (which is an upper bound on choosing a noise value that violates ϵ-differential privacy). Our deployment provides privacy according to the daily action bounds shown in Table 5, which are all based on circuit counts since that is what our deployment will measure; users whose actions stay below these bounds will be protected under differential privacy. We protect users who use 90 or fewer general-purpose circuits per day, which could be used to access one site every ten minutes for 8 hours plus 10 additional circuits. We protect users who use 48 or fewer rendezvous circuits per day when not distinguishing between client-side or server-side, and otherwise 24 or fewer each of client-side and server-side rendezvous circuits per day: 75 percent of the onion sites we crawled (section 6) used 4 or fewer circuits, and so the number of circuits we protect could be used to access 1 onion site every 10 minutes for one hour.

Measurement Rounds

We ran three different 24-hour long measurement rounds. The first round of measurements was used to calibrate the noise added to our counters. We used previously published measurements of Tor activity [19] to allocate our privacy budget across the configured counters. We then measured general and onion service circuit usage from different relay positions to obtain updated estimates

[12] All framework components are available at `https://github.com/onionpop`.

of circuit activity, which we used to adjust the allocation of our privacy budget in the subsequent rounds.

In the second measurement round, we focused on measuring the number of direct connections from the SNS of interest to our relays serving in the rendezvous position based on the IP address and autonomous system (AS) number of the SNS. This was possible because the SNS runs a *single onion service* that connects directly to the rendezvous point rather than a normal onion service which builds a three-hop circuit to connect. This set of measurements allow us to verify our circuit purpose and position classifiers.

In the third measurement round, we enabled our classifiers and focused on counting the results of the predictions. We also configured a crawler under our control to access the SNS in order to assert that our deployment was working properly and to cross check our prediction results (the prediction results for our crawler's circuits were kept separate from the results for other circuits). During round three only, we configured a circuit sample rate of 0.12 and excluded our exit relays from the measurement (since they would not contribute to the middle relay prediction counters) in order to prevent resource bottlenecks in our deployment pipeline. The percentage of Tor network bandwidth that the relays in our deployment controlled during each measurement round is shown in Table 6.

Table 6: Combined positional relay bandwidth by percent for PrivCount deployment.

Round	Guard	Middle	Exit	Intro.	Rend.
Measurement 1	1.15%	0.78%	3.52%	0.88%	0.78%
Measurement 2	1.30%	0.87%	3.11%	0.99%	0.87%
Measurement 3*	1.03%	0.68%	*0.0%	0.77%	0.68%

* To mitigate potential resource issues, exit relays were excluded from measurement 3 (the classification round) since they would not have contributed to middle relay onion service prediction counters.

7.4 Research ethics and user safety

Our measurement study explicitly prioritizes user safety as a primary goal. We practice data minimization, limit measurement granularity, and provide additional security to the measurement process as described above. We

have incorporated feedback from the Tor Research Safety Board[13] into our methodology: on suggestion of the board we created a website explaining our study[14] and linked our measurement relays to it, and we informed the SNS of our intentions to measure their site (although we did not receive a response from any of the employees of the SNS).

Because the main classification-based measurements are done from middle relay positions, onion-encryption technically prevents us from learning any client-identifying information. Although this protects users to some extent, we further protect users by utilizing the state-of-the-art in safe Tor measurement tools and techniques. Specifically, we use PrivCount and the techniques set out by Jansen and Johnson [19] and Elahi *et al.* [11] to provide differential privacy and securely aggregate measurements across all of our relay data collectors.

The PrivCount counters are initiated to noisy values to ensure differential privacy is maintained, and are then blinded and distributed across several share keepers to provide a secure aggregation process. At the end of the process, we learn only the value of these noisy counts aggregated across all data collectors, and nothing else about the information that was used during the measurement process. Specifically, we do not learn relay-specific inputs to the final counter value, and client usage of Tor during our measurement is protected under differential privacy.

Importantly, we chose to show our proof-of-concept by only predicting accesses to a single onion site that we believed had non-trivial usage and that already has implied that it does not require anonymity by running a non-anonymous single onion service. We explicitly chose *not* to measure additional regular onion sites because: (i) we did not believe it was necessary to show the effectiveness of our techniques; (ii) we wanted to avoid leaking more information than necessary about specific onion site usage; and (iii) running a hidden onion service would imply that anonymity is required or at least desired by the service.

7.5 Results

In addition to measuring the results of our classifiers, we also focused our PrivCount deployment on direct measurements that would allow us to validate our classification results.

[13] `https://research.torproject.org/safetyboard.html`
[14] `https://onionpop.github.io`

Direct Measurements

The direct measurement results are shown in Table 7. We measured the number of observed circuits on our relays from the circuit entry, middle, and end (including various types of rendezvous circuits). Our measurements indicate a significantly lower number of onion service rendezvous circuits compared to non-onion service circuits, as expected. While we discuss how these measurements give us an idea of popularity below, here we note that there are more than an order of magnitude fewer rendezvous circuits compared to non-onion service circuits.

Table 7: Results for direct measurement of Onion Service protocol.

Circuit Count Description	Count ± 95% CI
Entry	20,351,667 ± 3.45%
Middle	16,212,157 ± 4.33%
End (Exit + Rendezvous + etc.)	18,904,815 ± 3.71%
Rendezvous (Client or Service)	272,180 ± 5.15%
Rendezvous Client	136,191 ± 5.15%
Rendezvous Service	136,874 ± 5.12%
Rendezvous Service to SNS ASN	718 ± 91.64%
Exit + Rendezvous Client	11,327,103 ± 6.19%
Exit + Rendezvous Service	11,394,600 ± 6.16%

Because there are many circuits built in Tor over the period of a day, the relative accuracy of our direct measurements is quite high: most of the 95% confidence intervals lie between 3 and 6 percent. The one outlier is the direct measurement from the rendezvous node position of connections from the SNS ASN, which we can use to measure its popularity since this particular SNS runs a single onion service. The confidence interval is higher than expected (91.64%) which indicates that the SNS onion service is much less popular than expected, with potentially fewer than one hundred accesses through our relays during our measurement period.

Classifier Measurements

A primary purpose of our measurement is to test the ability of our classifiers to detect when a relay serves on a rendezvous circuit, in the R-C-M1 position, and if

Table 8: Results for measurement of Onion Service classifier detection.

Classifier	# Positives	# Negatives
Purpose is Rendezvous*	114,762±28.54%	2,444,166±79.82%
Ground Truth Tests**	645 (100%)	0 (0%)
Position is R-C-M1*	49,679±32.99%	68,022±48.15%
Ground Truth Tests**	623 (96.5%)	22 (3.4%)
Site is SNS*	10±1200%	45,376±36.12%
Ground Truth Tests**	374 (60.0%)	249 (40.0%)

* These values may appear lower than expected because we sampled circuits at a rate of 12% due to resource constraints.

** The ground truth tests were run with a crawler accessing the SNS during measurement, so these values represent true positives and false negatives.

it can identify accesses to a site of interest (SNS in our case). To do this, we send circuit meta-data (including cell meta-data for cells transferred on the circuit) to our classifiers when the circuit ends and record the detection results. We also run a crawler that creates rendezvous circuits through our middle relays during our measurement. The circuits created by our crawler provide ground truth that we can use to evaluate the classifiers' true positive and false negative rates.

Our classifier detection measurement results (including our ground truth crawler tests) are shown in Table 8. We again see a similar trend in that an order of magnitude fewer rendezvous circuits are detected compared to non-rendezvous circuits. With these measurements, there is a significant amount of noise associated with our measurements; this is primarily because we added the full amount of noise to provide differential privacy while at the same time sampling only 12% of circuits due to resource constraints. This has significantly increased the relative noise in our measurements. As in our direct measurements, the low number of SNS circuits has also caused our measure of the number of positive SNS detections to appear insignificant due to the large confidence interval associated with the noise that we added in order to protect privacy.

Our ground truth measurements show that the true positive rate for the purpose classifier was 100%, the true positive rate for the position classifier was 96.5% while the false negative rate was 3.4%, and the true positive and false negative rates for the SNS classifier were 60% and 40%, respectively. With these results, we are optimistic that our classifiers are functioning as intended. We assert

that an adversary who is not concerned with privacy (and does not add noise) would be able to make much more precise measurements than we describe here.

Popularity

We estimate the popularity of the onion service protocol by computing the fraction of middle relay circuits that are rendezvous circuits. Middle relays that do not serve as the rendezvous point on a circuit cannot determine with certainty whether or not the circuit is a rendezvous circuit, but they can predict it by running our circuit purpose classifier. As previously discussed, we also measure the popularity of the onion service protocol independently and directly when our relays do serve as rendezvous points, since in that case our relays can distinguish client-side and server-side rendezvous circuits from others.

Table 9: Likely Onion Service popularity by fractions of circuits of various types.

Description	Method	Popularity
Onion Service Popularity (as % of non-onion circuits)		
Rendezvous / Entry	Direct	1.34%
Rendezvous / End	Direct	1.45%
Rendezvous Client / Exit + Rendezvous Client	Direct	1.20%
Rendezvous Service / Exit + Rendezvous Service	Direct	1.20%
Purpose is Rendezvous / Total	Classified	4.48%
SNS Popularity (as % of onion circuits)		
Rend. Service to SNS ASN / Rend. Service	Direct	0.52%
Site is SNS / Total	Classified	0.02%

Our popularity estimates are shown in Table 9. The entries in the table show several ways one could estimate popularity, with our classification-based estimates at the bottom of each section. The direct measurement approaches indicate that onion service popularity is between 1% and 1.5% based on circuit counts; for comparison, 0.9% of Tor traffic by volume (*i.e.*,, bytes) is onion service traffic (900 Mbit/s onion[15] of 100 Gbit/s total[16]) according to Tor metrics. Our classification-based estimate is a bit higher at 4.48%, but we note this result includes noise and an unknown number of false positives. Similarly, our direct measurement of accesses to the SNS onion site front-page is 0.52% of rendezvous service circuits whereas our classification-based estimate is 0.02%.

[15] `https://metrics.torproject.org/hidserv-rend-relayed-cells.html`
[16] `https://metrics.torproject.org/bandwidth.html`

7.6 Discussion

Our laboratory results from the previous sections show that WF at the middle relay position is just as effective w.r.t. recall and precision as has been shown from the guard position in previous works—both for closed- and open-world scenarios. On the other hand, our real-world results indicate that the base-rate of the site we chose (*i.e.*,, the SNS) was too low for our classifier to provide high confidence for its counter. However, what we *can* learn with high confidence is that the popularity of the SNS as an onion service is almost negligible when comparing SNS onion service circuits to all other onion service circuits. This result was unexpected: our intuition for picking this particular SNS was that it is known to be one of the most popular websites in the world. Our results show that a much lower FPR—up to two orders of magnitude lower—is necessary for WF to be useful in measuring individual onion sites.

8 Related work

Although there are many studies that explore the extent to which traffic analysis can leak information on Tor [12, 23, 24], here we focus on website and onion site fingerprinting.

8.1 Tor website fingerprinting attacks

The first WF attack on Tor was proposed and evaluated by Herrmann *et al.* and only achieved 3% success rate [15]. This research area has since seen great activity and the latest WF studies achieve more than 90% accuracy under specific conditions and scenarios [4, 14, 26, 28–30].

Recent work attempted to address WF on non-onion websites in an *open world* model and increases the scalability of evaluation approaches [26], but the *closed world* model has been considered realistic for the evaluation of WF on onion services [6] due to their limited number. It has been shown that a local and passive adversary can effectively detect onion service visits using circuit fingerprinting, and then apply website fingerprinting methods to infer to which website they belong [22]. Errors when classifying onion service websites have been explored in order to further improve WF techniques [25], but the practicality of monitoring a realistic number of sites even in the smaller onion service world is still in question [27].

To the best of our knowledge, we are the first to apply circuit, position, and WF techniques from middle relays, and we are the first to use our classification techniques on traffic initiated from real Tor users. While we apply our techniques for measurement purposes, recent work has shown how our techniques can be used to further target specific users [17].

8.2 Tor website fingerprinting defenses

Several defenses have been designed to mitigate WF attacks. Most of these defenses are based on link padding [10, 21, 29], that is, adding dummy messages that are indistinguishable from real ones in order to make the features that WF exploit ineffective. Prior work assumes that the middle collaborates in the defense and removes the padding, in which case our techniques would not be affected. End-to-end padding that does not depend on collaborating Tor infrastructure [6] could disrupt the traffic analysis techniques we leverage, but would come at a prohibitively-high performance cost.

Restricted routing—*e.g.*,, if middles were chosen and used long term as is currently the case with guards—would limit the number of users from which a middle could observe circuits. In that case, middles could lose much of the advantage over guards as preferential observation points.

8.3 Onion site enumeration

Existing HSDir lookup protocols have been shown to be vulnerable to attack by an adversary running low-bandwidth relays [3]. By exploiting the lookup protocols, an adversary running an HSDir can directly measure the popularity of the onion services whose addresses are assigned to it. Previous work has used this approach to better understand the popularity of content in the onion service ecosystem [2]. These attacks can be mitigated by changes in the HSDir protocol.

Tor is currently deploying next-generation onion services in order to limit the effectiveness of onion service enumeration attacks, but the planned defenses will not significantly change the flow of cells through a circuit (like padding does) and therefore we believe that they will not significantly affect the accuracy of our techniques.

9 Conclusion

We have shown that a significant amount of information is leaked to middle relay positions, although the extent of this threat is often overlooked. We describe how the design of Tor admits to middle relays a wider visibility over all users of the network because clients pick new middle relays for every circuit that they build. We have shown through extensive data collection and experimentation that traffic analysis techniques are as effective from internal middle positions as they are from ingress and egress (guard and exit) positions. In particular, we have built a traffic analysis pipeline that can detect a relay's position in a circuit, the purpose of the circuit, and identifies the onion service being accessed through a circuit. We have then put the pipeline into practice to measure the popularity of a well-known social network onion service: we are the first to apply these traffic analysis techniques on real Tor user traffic to the best of our knowledge. Although our measurement results are constrained in scale and accuracy due to resource and ethical concerns (constraints not shared by malicious actors), our framework provides the means to study effective mitigation to potential threats and to gather additional measurements.

9.1 Lessons learned

It is clear that more progress needs to be made and this present work provides positive first steps in that direction. We anticipate that classification techniques at middle relay positions will not deteriorate and point out some of the challenges to deploying them in the real-world. First, our pipeline was created in order to reduce the number of circuits that need to be processed by the WF classifier; we filter out real user circuits for training and non-onion service circuits with the circuit classifier during testing. This greatly reduced the overhead both in training and testing and improved our results. Therefore, careful filtering and data pre-processing are keys to successful real-world deployments. Second, our measurement was done in real-time: everything was kept in RAM, and we used a low circuit sampling rate of 0.12 due to computational and memory limitations. We found that real-world scale may overwhelm available resources and pragmatic compromises may need to be made. Third, we were very concerned with user safety in our real-world measurements and hence our results are noisy. Depending on the use-case (*e.g.*,, a malicious actor), noise may not be necessary; removing this requirement would reduce the operational overhead of running the privacy-preserving apparatus and may allow higher sampling rates.

9.2 Future work

Using our current WF classification pipeline, an adversary could target the guards that originate connections to websites of interest (*e.g.*,, the SNS). We have shown that there are a small number of SNS circuits, and therefore the set of guards used to access the SNS would also be small. An adversary could reduce the time and cost of a targeting attack by focusing on only these guards rather than, *e.g.*,, compromising guards at random and waiting until it is used to access a website of interest. Some related target attacks that depend on our techniques have recently been explored [17].

An alternative to compromising guards that route interesting connections is locating the originating client or destination onion service using middle relay network latency measurements. Hopper *et al.* [16] show the effectiveness of such attacks from malicious websites. Mapping latency between an adversarial middle and all Tor relays (or at least the most popular) [5] would assist in narrowing the network and geographic location of circuit originators (*e.g.*,, to a region or possibly a country).

An adversary could fingerprint protocols instead of websites to target a broader base of users. For example, a censoring regime may fingerprint Tor's pluggable transports (PT) from the middle relay positions. Fingerprinting PTs from the client-side—which is the current state-of-the-art—has a high FPR since PTs are designed to be confused with other protocols that the censor is reluctant to block. In contrast, fingerprinting PTs at a middle does not provide this same confusion since only Tor traffic is present in the Tor network and the protocols that the censor is reluctant to block (*e.g.*,, HTTPS) are not present. Assuming that the censor already has the ability to identify users on the client-side, the censor could greatly reduce the incidence of false positives in detecting PT circuits. Furthermore, using timing correlations between client-side observations could also identify PT users, and an adversary could use our fingerprinting techniques to identify which websites PT users visit.

References

[1] TC: A Tor control protocol (Version 1). `https://gitweb.torproject.org/torspec.git/tree/control-spec.txt`.

[2] Alex Biryukov, Ivan Pustogarov, Fabrice Thill, and Ralf-Philipp Weinmann. Content and popularity analysis of Tor hidden services. In *International Conference on Distributed Computing Systems Workshops*, 2014.

[3] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization. In *IEEE Symposium on Security and Privacy (S&P)*, 2013.

[4] 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.

[5] Frank Cangialosi, Dave Levin, and Neil Spring. Ting: Measuring and exploiting latencies between all tor nodes. In *Internet Measurement Conference*, 2015.

[6] Giovanni Cherubin, Jamie Hayes, and Marc Juarez. "Website fingerprinting defenses at the application layer". In *Proceedings on Privacy Enhancing Technologies (PoETS)*, pages 168–185. De Gruyter, 2017.

[7] Roger Dingledine, Nicholas Hopper, George Kadianakis, and Nick Mathewson. One fast guard for life (or 9 months). In *Workshop on Hot Topics in Privacy Enhancing Technologies*, 2014.

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

[9] Cytnhia Dwork. Differential privacy. In *International Colloquium on Automata, Languages and Programming*, 2006.

[10] 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 *IEEE Symposium on Security and Privacy (S&P)*, pages 332–346. IEEE, 2012.

[11] Tariq Elahi, George Danezis, and Ian Goldberg. Privex: Private collection of traffic statistics for anonymous communication networks. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1068–1079. ACM, 2014.

[12] Nathan S Evans, Roger Dingledine, and Christian Grothoff. A practical congestion attack on tor using long paths. In *USENIX Security Symposium*, 2009.

[13] David Goulet, Aaron Johnson, George Kadianakis, and Karsten Loesing. Hidden-service statistics reported by relays. Technical report, Tor Project, April 2015.

[14] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *USENIX Security Symposium*, pages 1–17. USENIX Association, 2016.

[15] 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.

[16] Nicholas Hopper, Eugene Y Vasserman, and Eric Chan-Tin. How much anonymity does network latency leak? *Transactions on Information and System Security*, 13(2), 2010.

[17] Aaron D Jaggard and Paul Syverson. Onions in the Crosshairs: When The Man really is out to get you. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2017.

[18] Rob Jansen and Nicholas Hopper. Shadow: Running Tor in a box for accurate and efficient experimentation. In *Network & Distributed System Security Symposium (NDSS)*, 2012.

[19] Rob Jansen and Aaron Johnson. Safely measuring tor. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1553–1567. ACM, 2016.

[20] 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.

[21] 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 (ESORICS)*, pages 27–46. Springer, 2016.

[22] Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas. Circuit fingerprinting attacks: Passive deanonymization of tor hidden services. In *USENIX Security Symposium*, pages 287–302. USENIX Association, 2015.

[23] Prateek Mittal, Ahmed Khurshid, Joshua Juen, Matthew Caesar, and Nikita Borisov. Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting. In *ACM Conference on Computer and Communications Security (CCS)*, 2011.

[24] S.J. Murdoch and G. Danezis. Low-Cost Traffic Analysis of Tor. In *IEEE Symposium on Security and Privacy (S&P)*, pages 183–195. IEEE, 2005.

[25] 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.

[26] Andriy Panchenko, Fabian Lanze, Andreas Zinnen, Martin Henze, Jan Pennekamp, Klaus Wehrle, and Thomas Engel. Website fingerprinting at internet scale. In *Network & Distributed System Security Symposium (NDSS)*, pages 1–15. IEEE Computer Society, 2016.

[27] Andriy Panchenko, Asya Mitseva, Martin Henze, Fabian Lanze, Klaus Wehrle, and Thomas Engel. Analysis of Fingerprinting Techniques for Tor Hidden Services. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2017.

[28] 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.

[29] 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.

[30] 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.

[31] Tao Wang and Ian Goldberg. On realistically attacking tor with website fingerprinting. In *Proceedings on Privacy Enhancing Technologies (PoETS)*, pages 21–36. De Gruyter Open, 2016.

Publication

Does encrypted DNS imply Privacy? A Traffic Analysis Perspective

Publication Data

Siby, S., Juarez, M., Diaz, C., Vallina-Rodriguez, N., and Troncoso, C. Does encrypted DNS imply privacy? A traffic analysis perspective. *Submitted to the USENIX Security Symposium* (2020)

Contributions

- Principal author. First and second authors contributed equally.

Does Encrypted DNS Impliy Privacy? A Traffic Analysis Perspective

Sandra Siby[1], Marc Juarez[2], Claudia Diaz[2], Narseo Vallina-Rodriguez[3,4], and Carmela Troncoso[1]

[1] EPFL SPRING Lab, Lausanne, Switzerland
[2] KU Leuven, ESAT/COSIC and iMinds, Leuven, Belgium
[3] IMDEA Networks, Madrid, Spain
[4] ICSI, Berkeley, US

Abstract. Virtually every connection to an Internet service is preceded by a DNS lookup. These lookups are performed in the clear without integrity protection, enabling manipulation, redirection, surveillance, and censorship. In parallel with standardization efforts that address these issues, large providers such as Google and Cloudflare are deploying solutions to encrypt lookups, such as DNS-over-TLS (DoT) and DNS-over-HTTPS (DoH). In this paper we examine whether encrypting DoH traffic can protect users from traffic analysis-based monitoring and censoring. We find that performing traffic analysis on DoH traces requires different features than those used to attack HTTPS or Tor traffic. We propose a new feature set tailored to the characteristics of DoH traffic. Our classifiers obtain an F1-score of 0.9 and 0.7 in closed and open world settings, respectively. We show that although factors such as location, resolver, platform, or client affect performance, they are far from completely deterring the attacks. We then study deployed countermeasures and show that, in contrast with web traffic, Tor effectively protects users. Specified defenses, however, still preserve patterns and leave some webs unprotected. Finally, we show that web censorship is still possible by analysing DoH traffic and discuss how to selectively block content with low collateral damage.

1 Introduction

The Domain Name System (DNS) is a critical subsystem of the Internet infrastructure, on which most Internet-applications depend. Only in the

first quarter of 2019, more than 5 trillion DNS messages were exchanged per month [8]. The vast majority of such messages are sent in the clear [25], exposing the destination of communications to a number of entities: Internet Service Providers (ISPs), Autonomous Systems (ASes), or state-level agencies, can monitor users' activities [32], hence enabling mass surveillance [18], and easing network censorship by filtering and redirecting DNS traffic [71, 73].

The lack of mechanisms to enhance DNS privacy raise serious concerns among advocates [6] and Internet governance and standardization bodies [24]. Among the solutions that have been proposed to prevent the inspection of domain names, two protocols have been standardized and deployed: DNS-over-TLS (DoT) [39] and DNS-over-HTTPS (DoH) [38]. These protocols protect the communication between the client and the recursive resolver. More specifically, DoH uses HTTP/2 over TLS, and thus, is well suited for encrypting browsing-related DNS lookups [41]. Companies such as Google and Cloudflare have launched public DoH resolvers [2, 11], and Mozilla recently added DoH support to Firefox [28].

Under the assumption that encryption is enough to provide lookup confidentiality, existing evaluations of DoH implementations have focused on understanding the impact of the underlying transport protocol and encryption on performance [58, 59]. Yet, it is known that traffic features such as volume and timing can reveal the destination of the communication [34, 49, 52, 55, 64, 70].

In this paper we perform, to the best of our knowledge, the first traffic analysis study of encrypted DNS from a security and privacy angle. We consider an adversary placed between the client and the DNS resolver that aims at identifying which web page is visited by users, to either perform surveillance on users' traffic or censor access to certain resources. We focus on the case of DoH, as its adoption by large industry actors makes it prevalent in the wild.

The particularities of DNS traffic make it resistant to traditional traffic analysis techniques [34, 49, 52, 55, 64, 70]. We identify a *novel set of features* based on n-grams that capture local characteristics of traces that enable successful traffic analysis for encrypted DNS. We show how this set of features is robust to changes in the environment (e.g., end-user location or evolution of pages over time) or in the client's configuration (e.g., choice of client application, platform or recursive DNS resolver) Furthermore, we find that our new feature set provides *comparable or better results* than the state-of-the-art in website fingerprinting.

Motivated by our exchange with Cloudflare after responsible disclosure, we evaluate existing traffic analysis defenses: the standardized EDNS0 padding [50] and the use of Tor [5]. We find that in our setup, contrary to what was suggested by Cloudflare engineers, EDNS padding strategies cannot completely deter our

attack. Also, as opposed to traditional web traffic fingerprinting in which Tor offers little protection against traffic analysis, in the case of DoH, Tor is an extremely effective defense.

Finally, we measure the potential of encryption to hinder current DNS-based censorship practices. Using a *novel information-theoretic model* we show that, given that the size of the domain names associated with the resources embedded in a webpage visit (*e.g.*, third-party services, or content-providers) are the primary source of information in DoH traffic, this information can be used by censors to maintain their practices without much impact on other traffic.

To summarize, our main contributions are as follows:

- We conduct the first study of the vulnerability of DoH traffic to traffic analysis attacks. We show that traditional web fingerprinting techniques do not work on DoH and propose a new feature set to capture local characteristics (Section 5.1).
- We show that traffic analysis is effective against DoH, achieving the same accuracy as regular web fingerprinting while requiring $5x$ less volume of data. We show that factors such as end-user location, choice of recursive DNS resolver, client-side application, or platform affect, but do not stop, the attacks (Section 5).
- We evaluate existing traffic analysis countermeasures and show that only Tor can fully protect DoH traces (Section 6).
- We propose an information-theoretic model to evaluate the feasibility of DNS-based censorship when DNS lookups are encrypted (Section 7).
- We gather the first dataset of encrypted DNS traffic collected in a wide range of environments (Section 4).[5]

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 [6]. 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

[5] Our dataset and code will be made public upon acceptance.

[6] Over time, other applications have been built on top of DNS [13, 16]

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 1 summarizes this process.

Enhancing DNS Privacy. As with other network protocols, security was not a major consideration in the first versions of DNS, and thus DNS traffic has been sent in the clear over (in some cases, untrusted) networks. Over the last few years, security and privacy concerns have fostered the appearance of solutions aiming to make DNS traffic resistant to eavesdropping and tampering.

Early efforts for enhancing DNS security include protocols such as DNSSEC [10] and DNSCrypt [9]. DNSSEC introduces digital signatures to prevent manipulation of DNS data. It does not, however, provide confidentiality. DNSCrypt, first deployed by OpenDNS, both encrypts and authenticates DNS traffic between the client and the recursive resolver. However, it was never proposed to the IETF for standardization so it did not achieve wide adoption.

The IETF approved DNS-over-TLS (DoT) [39] and DNS-over-HTTPS (DoH) [38] as Standards Track protocols in 2016 and 2018, respectively. In DoT, a DNS client establishes a TLS session with a recursive resolver (usually on port TCP:853 [39] as standardized by IANA) 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.

In DoH, the local DNS resolver establishes an HTTPS connection to the recursive resolver and encodes the DNS queries as HTTP requests. DoH considers the use of HTTP/2's Server Push mechanism. This enables the server to preemptively push DNS responses to clients that are likely to follow a DNS lookup [40], thus reducing communication latency. As opposed to DoT, which uses a dedicated TCP port for DNS traffic and thus it is easy to monitor and block, DoH lookups can be sent along non-DNS traffic using existing HTTPS connections (yet potentially blockable at the IP level). However, DoT may be more convenient for enterprise network administrators, as it allows keeping tighter control over the DNS traffic.

There are several available implementations of DoT and DoH. Cloudflare and Quad9 provide both DoH and DoT resolvers, Google supports DoH, and Android P (currently in beta version) has native support for both DoH and 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 plaintext DNS if the resolution over DoH fails. Cloudflare also

distributes a stand-alone DoH client and, in 2018, they released a hidden resolver that provides DNS over Tor, not only protecting lookups from eavesdroppers but also providing anonymity for clients towards the resolver. Other protocols, such as DNS-over-DTLS [60], an Experimental RFC proposed by Cisco in 2017, and DNS-over-QUIC [17], proposed to the IETF in 2017 by industry actors, are not widely deployed so far.

Several academic works study privacy issues related to DNS. Shulman suggests that encryption alone may not be sufficient to protect users [63]. Our results confirm her hypothesis that 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 [35]. Finally, Imana et al. study privacy leaks on traffic between recursive and authoritative resolvers [42]. This is not protected by DoH and it is out of scope of our study.

3 Problem statement

In this paper, we set to answer the question: is it possible to infer which websites a user visits from observing encrypted DNS traffic? This information is of interest to multiple actors, *e.g.*, entities computing statistics on Internet usage [4, 7], entities looking to identify malicious activities [3, 19, 71], entities performing surveillance [32, 33], or entities performing censorship [21, 57].

We consider an adversary that can collect traffic between the user and the DNS recursive resolver (red dotted lines in Figure 1), and thus can link lookups to a specific origin IP address. Such an adversary could be present on the users' local network, near the resolver, or anywhere along the path (e.g., an ISP or compromised network router).

Depending on her location, the adversary may or may not observe the subsequent HTTP connection to the destination host. For instance, an adversary could be located in an AS that lies between the user and the resolver —*e.g.*, when using third-party DNS resolvers like Quad9 rather than their ISP-provided one—, but not between the user and the destination host. We performed measurements from our university network to verify that this is the case in a non-negligible number of cases. Furthermore, BGP hijacking attacks, which are becoming increasingly frequent [1], 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 [32, 33].

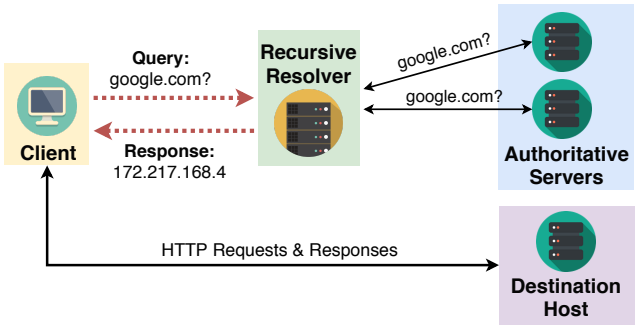


Figure 1: DNS resolution: To visit `www.google.com`, a user queries the recursive resolver for its IP. If the record is not cached, the recursive resolver queries an authoritative resolver and forwards the response to the client. The client uses the IP in the response to connect to the server via HTTP. We consider an adversary placed between the client and the resolver (i.e., observes the red dotted lines).

In the case of an adversary that also has access to the HTTP connection, one could argue that the subsequent HTTP(S) connection reveals visited domains even when encrypted. Fields such as the destination IP or the Server Name Indicator (SNI) may reveal the visited domain to the adversary in the case of TLS traffic. That could be further aggravated by HTTP flows emanating without encryption from the same user machine [18]. However, with the increasing prevalence of virtual hosting and Content Delivery Networks, and the implementation of protocols such as IPv6 and TLS 1.3, determining the destination domain of the connection without traffic analysis becomes more difficult. Thus, data leaked by encrypted DNS becomes even more relevant. While the adversary could perform traditional website fingerprinting, we show that fingerprinting DoH achieves the same accuracy while requiring less volume of data: our DoH traces are in average 5 times shorter in number of packets than HTTPS traces for web traffic.

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. A DNS trace, which we also call DoH trace, comprises the resolution of the visited website first-party domain, and the subsequent resolutions for the resources contained in the website, e.g., images, or 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 different adversaries depending on their goals: first, *monitoring* the browsing behavior of users, which we study in Section 5; and second *censoring* what pages users visit, which we study in Section 7. We note that there is a very important difference between these two goals regarding data collection. Monitoring does not require the adversary to take any action based on her observations. Thus, she can collect full traces to make their inferences as accurate as possible. In contrast, censorship adversaries need to find out which domain is being requested as fast as possible so as to interrupt the communication, so they must act on partial traces.

4 Data collection

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 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 top domains of the Alexa 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

To collect the traces we set up Ubuntu 16.04 virtual machines with DoH clients that send DNS queries to a public DoH resolver. We use Selenium[7] (version 3.14.1) to automatically launch a full-fledged browser and visit a webpage from our list and trigger the DNS lookups. We repeat this process for every webpage in the list restarting the browser every time to ensure that the cache and profile do not affect collection. We run *tcpdump* to capture the network traffic between the DoH client and the resolver. We filter the traffic by destination port and IP to obtain the final DoH trace.

To study the influence of various parameters on DoH traffic, we collect data in different scenarios varying end user location and platform, DoH client and resolver, and different DNS traffic analysis defenses. Table 1 provides an overview of the collected datasets. To better understand the vulnerabilities of DNS encryption we opted for having heterogenous experiments rather than in-depth studies of few cases, resulting in the difference in samples among the datasets. 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.

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

Table 1: Overview of datasets.

Name	Identifier	# webpages	# samples
Desktop (Location 1)	LOC1	1,500	200
Desktop (Location 2)	LOC2	1,500	60
Desktop (Location 3)	LOC3	1,500	60
Raspberry Pi	RPI	700	60
Firefox with Google resolver	GOOGLE	700	60
Firefox with Cloudflare resolver	CLOUD	700	60
Firefox with Cloudflare client	CL-FF	700	60
Open World	OW	5,000	3
DoH and web traffic	WEB	700	60
DNS over Tor	TOR	700	60
Pad only queries	EDNS0-REQ	700	60
Pad both queries and responses	EDNS0-FULL	700	60

Data curation. We curate the datasets to ensure that our results are not biased by spurious errors in collection, or website behaviors that are bound to generate classification errors unrelated to the characteristics of DNS traffic with respect to traffic analysis attacks.

Concretely, we aim at identifying two cases. First, the cases in which different domains generating the same exact DNS traces. These occur when webpages redirect to other pages or to the same resource, and when web servers return the same errors (e.g., 404 not found or 403 forbidden). Second, the case in which websites change during collection for reasons other than those variations due to their organic evolution. For instance, pages that go down during the collection period. When this happens, the captured traces do not represent the expected behavior of the page.

To identify these cases, 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. Then we conduct two checks. First, we look at the HTTP response status of the *top level domain*, i.e., the URL 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 solutions, forbidden responses due to geoblocking [51] (403), internal server errors (500), and so on. We mark these domains as conflicting. Second, we confirm that the top level domain 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 redirect 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 conflicting 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 conflicting to non-conflicting or vice versa). We study the effects of keeping and removing these conflicting webpages in Section 5.2.

5 Website fingerprinting through DNS

Website fingerprinting attacks enable a local eavesdropper to determine which pages a user is accessing over an encrypted or/and anonyimized channel. Website fingerprinting has been shown to be effective on HTTPS [31, 49, 52], OpenSSH tunnels [30, 48], encrypted web proxies [37, 66] and VPNs [36], and even on anonymous communications systems such as Tor [26, 34, 55, 56, 64, 68–70].

Website fingerprinting 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. These patterns can be indirectly observed, even if the traffic has been encrypted or anonymized.

Some of the patterns exploited by website fingerprinting are correlated with patterns in DNS traffic. For instance, which resources are loaded and their order, determines the order of the corresponding DNS queries. Thus, it is likely that website fingerprinting can also be done on DNS traffic encrypted with protocols such as DNS-over-HTTPS (DoH). In this paper we call *DNS fingerprinting* the use of traffic analysis to identify the web page that generated a trace of encrypted DNS traffic, 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 HTTPS traffic.

5.1 DNS traffic fingerprinting

As in website fingerprinting, we treat DNS fingerprinting as a supervised learning problem: the adversary first collects a training dataset of network traces for a set of pages, where the page (label) corresponding to a network trace is known. The adversary extracts features from the network traces (e.g., lengths of network packets) and trains a classifier to identify the page given a network trace. To deploy the attack, the adversary collects traffic from a target user and feeds it to the classifier to determine which page generated that traffic.

Traffic variability. In website fingerprinting, conditions such as networks conditions and embedded third-party advertisements, introduce variance in traffic traces sampled for the same website. Similarly, DNS traces also vary over time. Thus, the adversary must collect multiple samples for each page in order to obtain a robust representation of the page.

Some of this variability has similar origin to that of web traffic. For instance, the dynamic nature of websites that results on varying the DNS lookups associated with third-party embedded resources; 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); or the effects of content localization and personalization, which determines which resources are served depending on the location of the user, or her actions (e.g., logged in or not).

Additionally, there are some factors specific to DNS traffic. Concretely, the effect of the local resolver, which depending on the state of the cache may or may not launch requests to the authoritative server, resulting in different traffic patterns; or the DNS-level load-balancing (e.g., CDNs) which may provide different IPs for a resource [20].

Feature engineering. DNS traffic presents unique challenges with respect to web traffic for fingerprinting. Besides the extra traffic variability, DNS responses are smaller than web resources. In most cases, DNS requests and responses fit in one single TLS record, even if they are wrapped within HTTP requests like in DoH. These particularities hinder the use of traditional website fingerprinting features on DoH traffic.

As a matter of fact, in our preliminary experiments, we attempted to use features and techniques already used in the web traffic fingerprinting literature [34, 55]. Most of such features are based on aggregate metrics of traffic traces such as the total number of packets, total bytes, and their statistics (e.g., average, standard deviation). We found that these features are not as relevant for DoH traffic. For instance, the accuracy of the k-fingerprinting attack [34], which includes

most website fingerprinting features considered in the literature, drops from 95% to just 74% when applied on DoH traffic (see Table 4).

We present a novel feature set that is specifically designed for encrypted DNS traffic. The key idea is to represent traces as n-grams of TLS record lengths. The intuition is that n-grams capture patterns in request-response size pairs which are especially relevant for DoH, as TLS records often contain either a request or a response. To some extent, they also capture the local order of the length sequence. We take tuples of n consecutive TLS record lengths in the DoH traces trace and count the number of their occurrences in each trace. For instance, for the trace $(-64, 88, 33, -33)$, the uni-grams are $(-64), (88), (33), (-33)$ and the bi-grams are $(-64, 88), (88, 33), (33, -33)$. To the best of our knowledge, n-grams had 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 and thus are a good candidate feature for DoH traffic fingerprinting. 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 taking tuples burst lengths in the burst sequence. 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 types of features. We observed a marginal improvement in the classifier on using tri-grams at a substantial cost on the memory requirements of the classifier. We also experimented with the timing of packets but, as in website fingerprinting [69], we found it unreliable due to its dependence on the state of the network than on the content being served. Thus, they encode little information about the visited website. 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 decided to use Random Forests (RF), which have been demonstrated to be very effective for traffic analysis tasks [34, 43].

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. RFs mitigate this issue by randomizing the data and features over a large amount of trees, so that different subsets of features and data are used 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 of the trees as the aggregate function.

Validation. We evaluate the effectiveness of our classifier measuring the *Precision*, *Recall* and *F1-Score* (Appendix A.1) in two scenarios typically used in the web traffic analysis literature. A *closed world*, in which the adversary knows the set of all possible webpages that users may visit; and an *open-world*, in which the adversary only has access to a set of *monitored* sites, and the user may visit webpages outside of this set.

We use 10-fold cross-validation in all of our experiments to measure biases related to the overfitting of the classifier. Cross-validation is a standard methodology to evaluate overfitting in machine learning. In cross-validation, the samples of each class are divided in ten disjoint sets. The classifier is then trained on nine of the sets and tested in the remaining one, proving ten samples of the classifier performance on a set of samples on which it has not been trained on. This gives an idea of how the classifier generalizes to unseen examples.

5.2 Evaluating the n-grams features

In this section we evaluate the effectiveness of our n-grams based website fingerprinting attack on DNS traffic in the closed- and open-world scenarios, as well as on HTTPS traffic.

Closed world We first study a closed world setting in which the adversary knows the set of webpages visited by a user. Table 2 shows the classifier's performance on the LOC1 dataset. We observe that considering the 1,414 curated webpages (see Section 4) instead of all 1,500 webpages results in just a 1% performance increase. Thus, in the remaining experiments we use the complete dataset.

We notice that the Alexa ranking contains URLs that refer to regional versions of he same service. For example, `google.es` and `google.co.uk` both point to Google, but are considered as two separate webpages in our dataset. Even though our classifier often misclassifies these cases, from an adversary's point of view, they can be considered equivalent classes. The third row in the table shows that considering classifications within the equivalence class of a domain

Table 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

as a success results in a performance improvement of 3-4%. See Figures 11 and 13 in the Appendix for the confusion graphs of this evaluation.

As pointed out by prior work on website fingerprinting, average metrics can give an incomplete and biased view of the classification results [54]. This is because the classifier's performance may vary significantly between different individual classes. We observe that this is also the case in DoH. Figure 2 depicts individual classes in a 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, for some webpages the classifier obtains low Precision but high Recall (red dots on the right of the Precision scatterplot) and, conversely, there are pages with high Precision but low Recall (red dots on the right of the Recall scatterplot). The latter case is very relevant for privacy since, every time the adversary identifies one of these pages, she is absolutely sure her guess is correct. In censorship, for instance, this enables the censor to block with certainty.

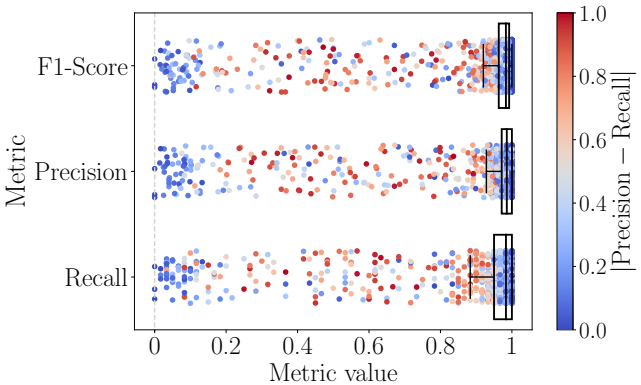


Figure 2: Performance per class in LOC1. Each dot represents a class and its color the absolute difference between Precision and Recall (blue low, red high).

Adversary's effort. To get an intuition about the data collection effort required by an adversary, we study the classifier's performance improvement with the number of samples used for training . We see in Table 3 that after 20 samples there are diminishing returns in increasing the number of samples per domain. To minimize the data collection effort, we collected 60 samples per domain for all our datasets except for the unmonitored websites in the open world, for which we collected three samples per domain.

Table 3: 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

We observe a difference with respect to prior work on web traffic analysis. Website fingerprinting studies in Tor report more than 10% increase between 10 and 20 samples [69] and between 2% and 10% between 100 and 200 samples [61, 64]. In DNS, we see a small increase between 10 and 20 samples, and a negligible difference after 20 samples.

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. One reason for this difference could be the presence of advertisements, which are an important source of intra-class variance in web traffic. They often change across visits, varying the sizes of the resources associated to the advertisement. However, the variance that advertisements add to DNS traffic might be more limited. Some publishers rely on ad-networks for ad mediation and, in some cases, the ad-network's domain and not the advertiser's will appear when fetching all the advertisements in the page [22].

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 [43, 45] 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 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 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. To perform cross-validation, we run 10 folds. To ensure that our classifier generalizes well to any unseen data in every fold, we consider a different combination of the monitored and unmonitored classes for training and testing.

To decide whether a target trace is monitored or unmonitored, we use a method proposed by Stolerman et al. [65]. 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 show in Figure 3, the average Precision-Recall ROC curve for the monitored class over 10 iterations varying the discrimination threshold, t, from 0 to 0.99 in steps of 0.1. We also show the random classifier, which indicates the probability of selecting the positive class uniformly at random, and acts as a baseline. We see that when $t = 0.8$, the classifier has an F1-score of ≈ 0.7. This result suggests that traffic analysis is a true threat to DNS privacy.

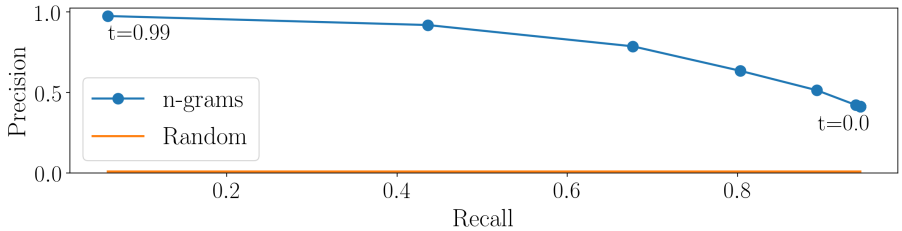


Figure 3: Precision-Recall ROC curve for open world classification, for the monitored class. The threshold, t, is varied from 0.0 to 0.99 in steps of 0.1 (standard deviation less than 1%).

Web traffic fingerprinting. Finally, we evaluate our n-grams features suitability for performing traditional web traffic fingerprinting. We compare them to the features set in the k-Fingerprinting attack, which includes a comprehensive set of features used in the website fingerprinting literature [34]. For the comparison we scaled down the closed-world, but fixed the same number of websites and samples per website between both feature sets. We used a random forest with the same parameters as classification algorithm in both cases.

We use the WEB dataset to evaluate the performance of the classifiers on only DoH traffic (DoH-only), only HTTPS traffic corresponding to web content traffic (Web-only), and no filter (DoH+Web). As shown in Table 4, not only the

n-grams achieve better performance than the k-Fingerprinting features on the DoH-only dataset but, surprisingly, they also outperform the k-Fingerprinting features on the Web-only and DoH-Web datasets.

Table 4: F1-Score of the n-grams and k-Fingerprinting features for different subsets of traffic: only DoH traffic (DoH-only), only HTTPS traffic corresponding to web traffic (Web-only) and and the full trace (DoH+Web).

	DoH-only	Web-only	DoH + Web
n-grams	0.87	0.99	0.88
k-Fingerprinting [34]	0.74	0.95	0.79

In both cases, an adversary who is able to intercept all communications, both with the resolver and the web server, can improve the success of the attack by adding web traffic, as shown by the increase in F1-Score between the first and the last rows. However, such an adversary is better off by discarding DoH traffic. We hypothesize that the added variability of DoH adds noise in small sites increasing the classifier errors.

5.3 DNS Fingerprinting Robustness

In practice, the capability of the adversary to distinguish websites is very dependent on environmental characteristics and differences in the setup while collecting data [44]. To understand the impact of the environment on DNS fingerprinting success we run 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). We now study how this variability impacts the performance of the classifier.

We consider collect data LOC1 for 10 weeks between 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 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 a significant drops – more than 10% drop in F1-score when the training and testing are separated 8 weeks.

Table 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, the adversary should 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 slight decrease in performance. Thus, a continuous collection strategy can suffice to maintain the adversary's performance without requiring large periodic 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 for load-balancing requests, *e.g.*, using anycast and EDNS [53, 62].

We collect data in three locations, two countries in Europe (LOC1 and LOC2) and a third in Asia (LOC3). Table 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 yield 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 on around 77% Precision and Recall, but training on LOC1 and testing on LOC2 yields 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 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

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 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

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

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

Table 6 (center-left) shows the result of the comparison. As expected, training and testing on the same resolver yields the best results. In particular, we note that even though Google hosts other services behind its resolver's IP and thus DoH traffic may be mixed with the visited website's traffic (e.g., if a web embeds Google third-party) the classifier performs equally for both resolvers.

As in the location setting, we observe an asymmetric decrease in one of the directions: training on GOOGLE dataset and attacking CLOUD results in 13%

F1-socre, while attacking GOOGLE with a classifier trained on CLOUD yields similar results as training on GOOGLE itself.

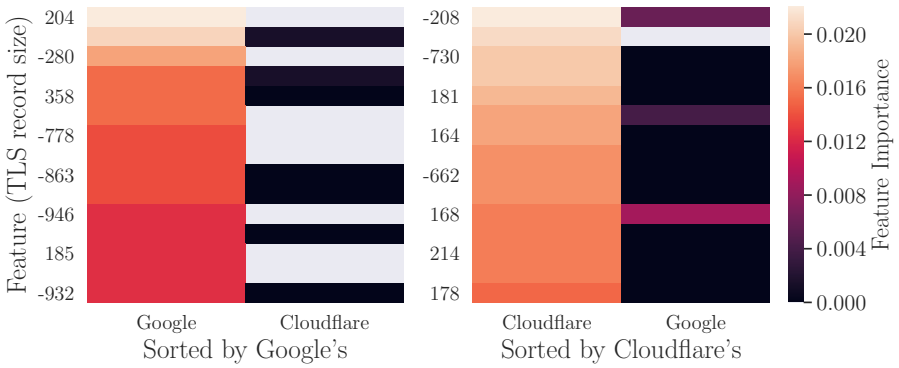


Figure 4: Top 15 most important features in Google's and Cloudflare's datasets. On the left, features are sorted by the results on Google's dataset and, on the right, by Cloudflare's.

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 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 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 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.

Aiming at understanding this drop, we take 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 (see Figure 10 in Appendix A.3). We repeated the cross classification after adding 8 bytes to all RPI TLS record sizes. 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.

Influence of DNS client. Finally, we consider different client setups: Firefox's trusted recursive resolver or TRR (CLOUD), Cloudlflare'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 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 a proxy to man-in-the-middle the DoH connection between the client and the resolver[8], obtaining the OpenSSL TLS session keys with Lekensteyn's scripts[9]. We use this proxy to decrypt DoH captures for Firefox configured to use Cloudflare's resolver, but we could not do the same for Google. Instead, we man-in-the-middle a curl-doh client[10], 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 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.

[8] https://github.com/facebookexperimental/doh-proxy
[9] https://git.lekensteyn.nl/peter/wireshark-notes
[10] https://github.com/curl/doh

Robustness Analysis Takeaways

The results in the previous section reveal that to obtain best results across different configurations the adversary would need to train a classifier for each targeted setting. Then, of course, she would need to be able to identify her victim's configuration. 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 the Server Name Indication (SNI), cipher suites ordering, etc.) [47]. Even if in the future these features are not available, we found that the characteristics of the 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.

Regarding users' platform, we see little difference between desktops, laptops, and servers in Amazon Web Services. Only when the devices are as different as a desktop and a constrained device the classifier's accuracy drops.

Finally, our longitudinal analysis reveals that, even though webs change over time these changes are not drastic. Therefore, it should not be hard for the adversary to keep up with the changes by continuously collecting samples and incorporating them to her training set.

Survivors and Easy Preys. We study whether there are websites that are particularly good or bad at evading fingerprinting under all the configurations evaluated in this section. We compute the mean F1-Score across all configurations as an aggregate measure of the attack's overall performance, and analyze the skew of its distribution on individual websites. We plot the CDF of the distribution of mean F1-scores over the websites in Figure 5. 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.

We looked into the tails of this distribution and ranked sites by lowest mean F1-Score and lowest standard deviation. On top of that ranking we have sites that *survived* the attack in all configurations. Among the survivors we found Google and errored sites that misclassify between each other. For other surviving sites, after manual inspection we did not find a pattern in the websites structure or the resource loads that explains why these sites survive. We leave a more in-depth analysis of the survival of these sites for future work. In Appendix A.5 we list the top-10 sites in the tails of the distribution.

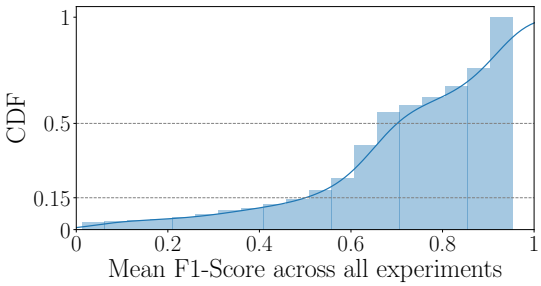


Figure 5: Cumulative Distribution Function (CDF) of the per-class mean F1-Score.

6 DNS defenses against fingerprinting

In this section, we compare existing defenses aimed at preventing traffic analysis attacks on encrypted DNS traces.

EDNS(0) Padding. EDNS (Extension mechanisms for DNS) is a specification to increase the functionality of the DNS protocol [27]. One of the options is the addition of *padding* [50] by both DNS clients and resolvers in order to prevent size-correlation attacks on encrypted DNS. The recommended padding policy is to pad DNS requests to the nearest multiple of 128 bytes and DNS responses to the nearest multiple of 468 bytes [15]. Cloudflare's DoH client provides functionality to set EDNS(0) padding to DNS queries, but leaves the specifics of the padding policy to the user. We modify the client source code to follow the recommended padding strategy. Google's specification also mentions EDNS padding. However, we could not find any option to activate this feature.

When conducting our data collection we discovered that Cloudflare's DoH resolver does *not* implement server-side padding[11]. To overcome this issue we set up an HTTPS proxy, *mitmproxy*, between the DoH client and the Cloudflare resolver. The proxy intercepts responses from Cloudflare's DoH resolver and pads them to the nearest multiple of 468 bytes.

Below we evaluate the effectiveness against traffic analysis of solely padding DNS queries (EDNS0-REQ), and padding both queries and responses (EDNS0-FULL).

Constant padding. To fully understand the potential of padding, we also simulate a setting in which *all* packets are padded to the same length (that of the longest

[11] We communicated this fact to Cloudflare. They replied on the 21st March that they would add it on the next release. To date this is still not implemented

packet in the dataset, with a size of 825 bytes). This implies that the classifier cannot exploit the TLS record size information.

DNS over Tor. We finally evaluate the use of Tor as a deterrent for traffic analysis attack. We use Cloudflare's DNS over Tor service as a target.

Results. Table 7 shows the classification results for all defenses. As expected, padding only DNS requests, while reducing the F1-score, is not as effective as padding both requests and responses. Padding both requests and responses, which was intended to alleviate traffic analysis attacks, is not as effective as expected. Padding all record sizes to the same size value greatly reduces the F1-score. However, it is not as effective as using Tor, probably because some order information of the records is still maintained, even if the size information is no longer available to the classifier.

The success of Tor for DNS encrypted traffic is a huge difference with respect to web traffic, where website fingerprinting obtains remarkable performance [34, 56, 64]. The reason is that DNS lookups and responses are fairly small, they result in mostly one or two Tor cells which in turn materialize in few observed TLS record sizes, making it difficult to find features unique to a page. We see a similar effect in the number of TLS records per trace – TOR traces are generally shorter and have less variance. Thus lengths-related features, which have been proven to be very important in website fingerprinting, are of no help in the DNS scenario. They only provide a weak 1% performance improvement. Web traffic contains much bigger resources and as a result TLS traces present more variability and are easier to fingerprint. much more information.

While DNS over Tor obtains the best results, when we look closely at the misclassified webpages, we find that webpages get misclassified within six clusters (see Figure 12 in the Appendix). We train a classifier considering all domains within a cluster as equivalent classes. This classifier achieves ≈55% accuracy, compared to 16% accuracy for random guessing. This means that despite Tor's protection the effective anonymity set for a webpage is much smaller than the total number of webpages in the dataset. We leave as future work a comprehensive analysis of what traffic characteristics contribute towards the formation of these clusters.

Finally, we evaluate the trade-off between the defenses' effectiveness and their communication overhead. To compute the overhead generated by each countermeasure, we collect 10 samples of 50 webpages with and without countermeasures.

Figure 6 shows the total volume distribution (sent and received data) for all cases. As expected, the EDNS0 padding (both REQ and FULL) add the least

Table 7: Classification results for countermeasures.

Method	Precision	Recall	F1-score
EDNS0-REQ	0.710 ± 0.005	0.700 ± 0.004	0.691 ± 0.004
EDNS-FULL	0.465 ± 0.007	0.460 ± 0.008	0.442 ± 0.007
Constant 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

overhead, but they also offer the least protection. DNS over Tor, in addition to being more effective than constant padding, also has a smaller overhead. We conclude that DNS repacketizing in addition to padding, as done in Tor, can be a promising avenue to explore.

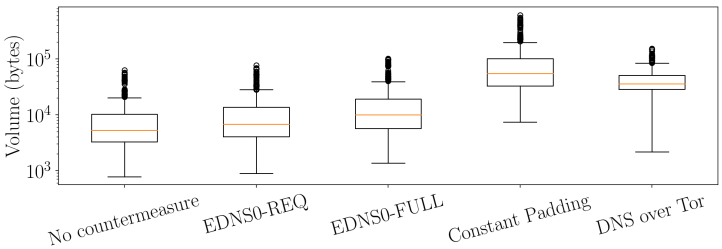


Figure 6: Total volume of traffic with and without countermeasures.

7 DNS encryption and censorship

DNS-based blocking is a wide-spread method of censoring access to web content. Censors inspect DNS lookups and when they detect a blacklisted domain, they either reset the connection or inject their own DNS response [67]. DoH encrypts DNS by default, rendering content-based DNS blocking ineffective. If censors want to continue restricting access to content by blocking DNS, DoH forces them to block the resolver's IP. While this would be very effective, some DoH resolvers, such as Google's, do not necessarily have a dedicated IP. Thus, blocking their IP causes collateral damage that may be too expensive for the censor.

In this section, we study whether DoH traffic is really an effective countermeasure to deter DNS-based censorship. A censor aims at blocking access to a number of blacklisted domains. To achieve this goal, the censor needs to identify the domain as soon as possible to prevent the user from downloading any content. We aim at answering two questions: how long must the adversary observe the

connection to uniquely identify the domain? Second, based on the answer to the first question, what strategy allows the censor to maximize censoring rates while minimizing collateral damage?

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 fulfill the censor's goal *the first packets* of the trace need to be unique. In the the following, we study the uniqueness of DoH traffic when only the l first TLS records (or packets, for short) have been observed.

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 and its sign represents the direction – negative for incoming (DNS to client) and positive otherwise.

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

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

where $H(W \mid S_l = o)$ is the Shannon entropy of the probability distribution $\Pr[W \mid S_l = o]$ describing the likelihood that the adversary guesses websites in W given the observation o. This entropy measures distinguishability of traces up to packet l. For instance, if every DoH trace started with a packet of a different size, then the entropy $H(W \mid S_1)$ would be 0, i.e., sites would be perfectly distinct from the first packet.

We show in Figure 7 the conditional entropy $H(W \mid S_l)$ for different number of webpages n in the LOC1 dataset. Every point is an average over 10 samples of n webs from the dataset selected uniformly at random with replacement. The shades represent the standard deviation across the 10 samples.

First, we observe that the conditional entropy decreases as the adversary observes more packets. 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 (reaching zero when $n = 10,100$ sites). We note that as we consider more websites, the likelihood of having two or more websites with identical traces increases. Thus, we observe a slower decay in entropy.

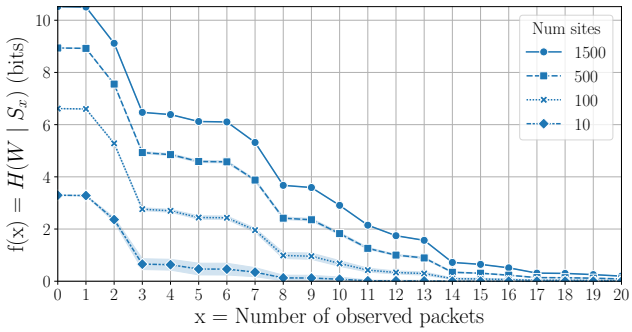


Figure 7: Conditional entropy $H(W \mid S_l)$ given partial observations of DoH traces for 10, 100, 500 and 1,500 webpages. Each data point is averaged over 10 samples.

A second observation is that the standard deviation is lower for small and large l's. The former is because the first packets correspond to the connection establishment. Thus, they are similar for all webpages. The latter is because as l increases, the traces become more dissimilar and thus the entropy is close to zero regardless of which websites are sampled. We also observe larger variation when few websites are considered. This is because we only have 1,500 webs. As the number of websites per group increases, there is more overlap among the groups used in the experiment. For 1,500 there is no variance because all samples contain the full dataset.

When considering all 1,500 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. The average trace length in our dataset is 96 packets. Thus, 15 packets is just 15% of the whole trace. This means that, 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.

Next, we investigate the cause behind the consistent entropy decrease within the first four TLS records. We hypothesized that it might be caused by the fact that one of these records contains the DoH query. Since the DoH protocol does not specify padding, uniqueness in the domain length would be directly observable in the trace. To verify our hypothesis we plot the frequency of the domain's and fourth record's length in Figure 8. We discarded TLS incoming packets – as they cannot contain a DoH query–, and TLS record sizes corresponding to HTTP2 control messages, e.g., the size "33" which corresponds to HTTP2 acknowledgements. We also removed outliers for sizes that occurred 5% or

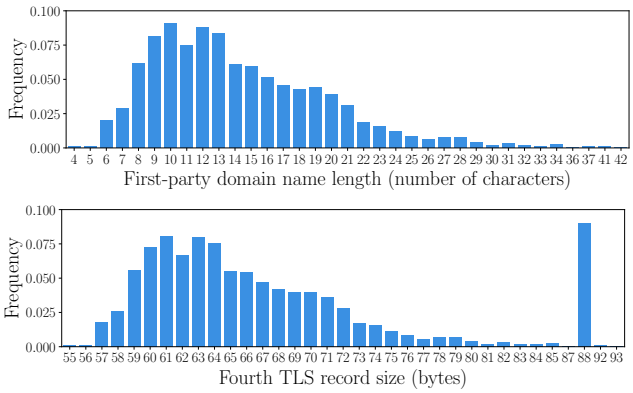


Figure 8: Histograms for domain name length (top) and fourth TLS record length (bottom) in the LOC1 dataset (normalized over the total sum of counts).

less times. We kept any size that could have contained a DoH query. For instance, we kept size "88" even though it appears too often to only be caused by DoH queries, as such packet size could be caused by queries containing 37-characters-long domain names.

The histogram of the sizes of the fourth TLS record in our dataset is almost identical to the histogram of domain name lengths. This confirms our hypothesis that the fourth packet often contains the first-party DoH query. We verified that the constant difference of 51 bytes between the two histograms is the size of the HTTPS header. We also observed that in some traces the DoH query is sent earlier, explaining the entropy decrease starting after the second packet.

7.2 Censor DNS-blocking strategy

Given that traces are not completely unique, the censor must act on guesses. When wrong, these guesses will cause collateral damage. Of course, the adversary can increase her confidence in her guesses by waiting to observe more TLS records. Thus, there is a trade-off between the collateral damage caused by erred guesses and the amount of content that can be accessed by users. We now discuss advantages and disadvantages of two strategies to censor a connection based solely on encrypted DNS traffic. We assume that upon decision, the censor uses standard techniques to block the connection [46].

High-confidence guesses. A possible strategy to minimize the likelihood of collateral damage is to act only upon seeing the entropy going lower than one bit. Following this strategy, the adversary would not block, on average, 15% of

the TLS records in the DoH connection. Those packets include the resolution to the first-party domain. Thus, 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). We note that quality degradation is a strategy already used in the wild as a stealthy form of censorship [46].

As a response to this censorship strategy, the client could just create a new connection for each DoH query so that the censor cannot distinguish DoH connections belonging to the censored webpage visit or to 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. An alternative strategy is to drop the DoH connection before the first DoH response arrives. This guarantees that the client cannot access any content, not even `index.html`. However, it implies that all domains that result on the same trace up to the first DoH query, i.e., all domains with same name length, would also be censored. We illustrate this effect in Figure 9, where we show the entropy decrease for different pairs of sites. We see that sites with different lengths (`facebook.com` and `nytimes.com`) are distinguishable on the fourth packet. However, when domains have the same name length the entropy only drops after the the DoH response, which is different per domain, and hence distinguishable. We note that, even for cases when the same service has different domain names with equal length (e.g., `google.es` and `google.be`) the entropy eventually drops to zero. For these cases instead of waiting, the adversary can also combine all equivalent pages in the same class, which as shown in Section 5.2 increases the performance of the classifier.

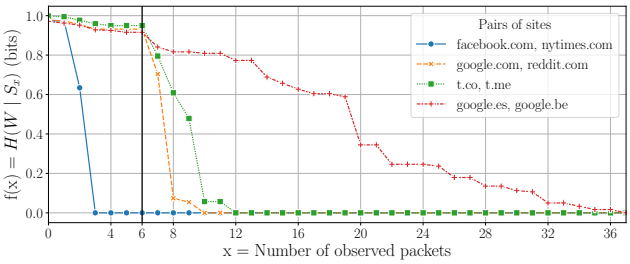


Figure 9: Conditional entropy over the number of observed packets for pairs of sites. The black bold vertical line corresponds to the median position of the first incoming packet, likely to contain the first DoH response.

Finally, we quantify the collateral damage incurred when blocking the first DoH query. The histogram in Figure 8 (top) represents the anonymity sets of websites with same domain name length. For instance, when blocking `nytimes.com`, that

has length 11, one would also block other 111 websites. In our data, anonymity set sizes are unevenly distributed. Only two websites have anonymity set one, and thus can be blocked with no collateral damage. We also observe that popular domains (according to the Alexa rank on the March 26) tend to have more common domain name lengths. The Pearson correlation coefficient between the domain name length and its Alexa rank for the top 1,000 domains is 0.49, which indicates a moderate-to-high correlation. In particular, the first top-five domains all lie in the 9-13 name length range, the most popular lengths. This is because these lengths correspond to the average length of a word in English and are the easiest to remember. Also, less popular domains often have a second- or third-level domain name such as tumblr or Wordpress sites.

Internet traffic volume distribution over domains follows a power-law [29], i.e., the Alexa top domains accumulate a large fraction of the overall internet traffic. Thus, blocking those domains not only has large collateral damage in terms of number of webs, but also traffic volume. On the contrary, blacklisting unpopular domains with uncommon lengths (in our dataset shorter than 8 or longer than 20 characters), not only blocks less websites, but also affect less overall traffic. The correlation between name length and popularity deserves a deeper study, since we show it is advantageous for some types of censors that tackle non-popular domains such as sites trading drugs.

8 Conclusions

We have performed the first evaluation of DNS-over-HTTPS vulnerability to traffic analysis. We have proposed a new set of features that characterize local patterns in traces. We show that these features are also suitable for web traffic fingerprinting, obtaining results comparable to the state of the art classifiers on HTTPS.

Our experiments show that, encryption is not sufficient to protect users from surveillance or DNS-based censorship. We also demonstrated that changes in factors such as end-user location, local DNS resolver, or client's platform negatively impact the attack performance, but in many cases traffic analysis is still pretty effective. Furthermore, it is easy for the adversary to recognize the setting of her target and select the most adequate classifier.

In terms of defenses, we show that the recommended EDNS0 padding strategies do not hinder traffic analysis. Repacketizing and padding, as done in anonymous communications, is required to defeat traffic analysis. We hope that these results serve to influence the evolution of standards on DNS privacy, and prompt main providers to prioritize the addition of countermeasures in their next releases.

This seems nowadays out of their plans [12, 14] even though they claim to strive for providing privacy.

References

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

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

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

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

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

[6] DNS Privacy - Current Work. https://dnsprivacy.org/wiki/display/DP/DNS+Privacy+-+Current+Work. Accessed: 2018-12-26.

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

[8] DNS Trends and Traffic. https://www.akamai.com/us/en/why-akamai/dns-trends-and-traffic.jsp. Accessed: 2018-12-26.

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

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

[11] Google DNS-over-HTTPS. https://developers.google.com/speed/public-dns/docs/dns-over-https. Accessed: 2018-05-07.

[12] Google Public DNS position on DNS-over-HTTPS (DoH). https://mailarchive.ietf.org/arch/msg/dnsop/GE8v2Yz6zsl28clDvlshGh3rYlc. Accessed: 2019-05-13.

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

[14] Mozilla's plans re: DoH. https://mailarchive.ietf.org/arch/msg/doh/po6GCAJ52BAKuyL-dZiU91v6hLw. Accessed: 2019-05-13.

[15] Padding Policies for Extension Mechanisms for DNS (EDNS(0)). https://tools.ietf.org/html/rfc8467. Accessed: 2019-05-10.

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

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

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

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

[20] 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.

[21] Anonymous. The collateral damage of internet censorship by dns injection. *SIGCOMM Comput. Commun. Rev.*, 42(3):21–27, 2012.

[22] Muhammad Ahmad Bashir and Christo Wilson. Diffusion of user tracking data in the online advertising ecosystem. 2018.

[23] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[24] S. Bortzmeyer. DNS Privacy Considerations. RFC 7626, 2015.

[25] Stephane Bortzmeyer. DNS privacy considerations. 2015.

[26] 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.

[27] J. Damas, M. Graff, and P. Vixie. Extension mechanisms for dns (edns(0)). RFC 6891, RFC Editor, April 2013.

[28] 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.

[29] Luca Deri, Simone Mainardi, Maurizio Martinelli, and Enrico Gregori. Graph theoretical models of dns traffic. In *9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 1162–1167. IEEE, 2013.

[30] 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 *IEEE Symposium on Security and Privacy (S&P)*, pages 332–346. IEEE, 2012.

[31] 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.

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

[33] Saikat Guha and Paul Francis. Identity trail: Covert surveillance using DNS. In *Privacy Enhancing Technologies Symposium (PETS)*. Springer, 2007.

[34] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *USENIX Security Symposium*, pages 1–17. USENIX Association, 2016.

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

[36] 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.

[37] Andrew Hintz. Fingerprinting websites using traffic analysis. In *Privacy Enhancing Technologies Symposium (PETS)*, pages 171–178. Springer, 2003.

[38] P. Hoffman and P. McManus. Dns queries over https (doh). RFC 8484, RFC Editor, October 2018.

[39] 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.

[40] G. Huston. DOH! DNS over HTTPS explained. `https://blog.apnic.net/2018/10/12/doh-dns-over-https-explained/`. Accessed: 2018-12-26.

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

[42] Basileal Imana, Aleksandra Korolova, and John S. Heidemann. Enumerating Privacy Leaks in DNS Data Collected above the Recursive. 2017.

[43] 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.

[44] 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.

[45] 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 (ESORICS)*, pages 27–46. Springer, 2016.

[46] Sheharbano Khattak, Tariq Elahi, Laurent Simon, Colleen M Swanson, Steven J Murdoch, and Ian Goldberg. SoK: Making sense of censorship resistance systems. *Proceedings on Privacy Enhancing Technologies (PoETS)*, 2016(4):37–61, 2016.

[47] 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.

[48] 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.

[49] Xiapu Luo, Peng Zhou, Edmond W. W. Chan, Wenke Lee, Rocky K. C. Chang, and Roberto Perdisci. HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows. In *Network & Distributed System Security Symposium (NDSS)*. IEEE Computer Society, 2011.

[50] A. Mayrhofer. The edns(0) padding option. RFC 7830, RFC Editor, May 2016.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] Andriy Panchenko, Fabian Lanze, Andreas Zinnen, Martin Henze, Jan Pennekamp, Klaus Wehrle, and Thomas Engel. Website fingerprinting at internet scale. In *Network & Distributed System Security Symposium (NDSS)*, pages 1–15. IEEE Computer Society, 2016.

[56] 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.

[57] 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.

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

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

[60] T. Reddy, D. Wing, and P. Patil. Dns over datagram transport layer security (dtls). RFC 8094, RFC Editor, February 2017.

[61] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. In *Network & Distributed System Security Symposium (NDSS)*. Internet Society, 2018.

[62] 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.

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

[64] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1928–1943. ACM, 2018.

[65] Ariel Stolerman, Rebekah Overdorf, Sadia Afroz, and Rachel Greenstadt. Classify, but verify: Breaking the closed-world assumption in stylometric authorship attribution. In *IFIP Working Group 11.9 on Digital Forensics*. IFIP, Springer, 2014.

[66] 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.

[67] 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.

[68] 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.

[69] 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.

[70] Tao Wang and Ian Goldberg. On realistically attacking tor with website fingerprinting. In *Proceedings on Privacy Enhancing Technologies (PoETS)*, pages 21–36. De Gruyter Open, 2016.

[71] Nicholas Weaver, Christian Kreibich, and Vern Paxson. Redirecting dns for ads and profit. In *FOCI*, 2011.

[72] 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.

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

A Appendices

A.1 Performance 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.2 Estimation of probabilities

In this section we explain how we have estiamted the probabilities for for the entropy analysis in Sections 5 and 7.

Basic definitions

A *traffic trace* or just *trace* is a finite 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 packet in the traffic trace. In an abuse of notation, we may drop the subindices and denote a traffic trace simply as s.

We assume there is a fix closed-world of m websites: $w_1, \ldots, w_m$. For each website w_i, we have sampled k_i traces, following the methodology described in this paper. We use set notation to denote that: $w_i = \{s_{i,1}, \ldots, s_{i,k_i}\}$, where $s_{i,j}$ is the j-th trace sampled for website w_i.

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.

Probability Space

We define sample spaces: $\Omega_W := \{w_i : i = 1, \ldots, m\}$ and $\Omega_S := \{s_{i,j} : i = 1, \ldots, m, j = k_1, \ldots, k_m\}$ and their corresponding random variables W and S, respectively.

As an example of a probability based on these two random variables, the probabilitiy of website w given that we have observed a trace s is: $P(W = w \mid S = o)$.

If we want to express probabilities of incomplete observations, e.g., we have observed up to l packets with $l < n$, we can define the sample space $\Omega_{S_l} := \{(s_i)_{i=1}^{l} \mid l < n\}$. Ω_{S_l} denotes samples of length-l subsequences, i.e., the first l packet sizes of traces in Ω_S. Let S_l be the corresponding random variable for Ω_{S_l}.

Then, we can express the probability for having observed l packets of a trace as: $P(W = w \mid S_l = o)$.

Conditional entropies

The probability $P(W = w \mid S = o)$ tell only how likely website w is if we have observed trace s. However, this probability may be similar for several sites – think of the case in which $A(s)$ contains those sites. In that case, the adversary may incur in false positives. We are interested in the distribution of $P(W \mid S = o)$ and its entropy $H(W \mid S = o)$. The entropy captures how much information the observation of s reveals about websites. For instance, in the case of similar probabilities for all sites, the entropy is going to be high, while if the probabilities for all sites except one are 0, it reaches its maximum – i.e., $A(s)$ contains one site.

In particular, the entropy $H(W \mid S_l)$ is interesting for a censor-type of adversary, who wants to drop connections for a blacklist of domains in real-time. This entropy decreases as $l \to n$ and thus we would like to know for which l the entropy is *sufficiently low*, according to a certain threshold, for the censor to take a decision. The smaller the l, the faster the censor can block the site. In particular, we are interested in the packet that contains the first DNS request as that would allow the censor to block the response for the first-party

domain. In addition, there is a trade-off between the threshold on the entropy and the collateral damage of blocking because an entropy $H(W \mid S_l)$ higher than zero means there are some chances of false positives and thus blocking of non-censored sites.

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,

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

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 allows us to simplify the Bayes rule formula since we can factor out $\Pr[W = w_i]$ in Equation 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 simply 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.3 Extra results on attack robustness

Table 8: Performance when training on the resolver indicated by the row and testing on the resolver indicated by the column (standard deviations less than 1%).

Precision	GOOGLE	CLOUD
GOOGLE	0.886	0.386
CLOUD	0.881	0.890

Recall	GOOGLE	CLOUD
GOOGLE	0.881	0.083
CLOUD	0.860	0.886

Table 9: Performance when training on the platform indicated by the row and testing on the platform indicated in the column (standard deviation less than 1% for same platform and less than 0.1% for cross-platform.

Precision	DESKTOP	RPI
DESKTOP	0.8848	0.0003
RPI	0.0003	0.8970

Recall	DESKTOP	RPI
DESKTOP	0.8816	0.0008
RPI	0.0010	0.8945

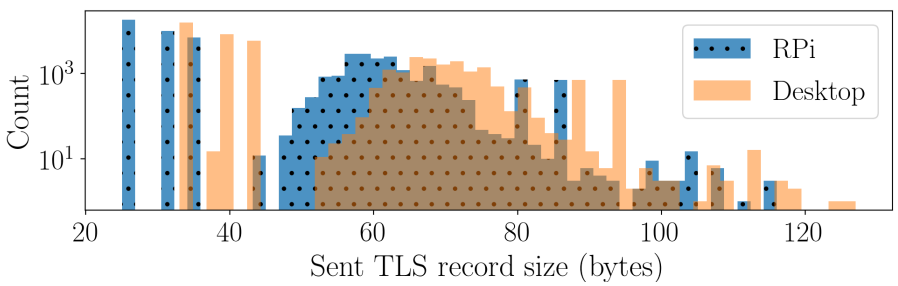


Figure 10: Distribution of user's sent TLS record sizes in platform experiment.

Table 10: 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

Table 11: Performance when training on the client setups indicated by the row and testing on the configuration indicated by the column (standard deviations less than 2%).

Precision	CLOUD	CL-FF	LOC2
CLOUD	0.890	0.646	0.000
CL-FF	0.257	0.896	0.089
LOC2	0.001	0.090	0.911

Recall	CLOUD	CL-FF	LOC2
CLOUD	0.886	0.267	0.001
CL-FF	0.080	0.893	0.073
LOC2	0.004	0.069	0.909

A.4 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 [54] and other classification tasks to understand classifier error [72].

Figures 11, 12 and 13 show the classification errors in the form of confusion graphs for some of the experiments presented in Section 5. The graphs were drawn using Gephi, a software for graph manipulation and visualization. Nodes in the graph are domains and edges represent misclassifications between domains. The edge source is the true label of the sample and the destination is the domain that the classifier confused it with. The direction of the edge is encoded clockwise in the curvature of the edge. Node size is proportional to the node's degree and nodes are colored according to the community they belong to, which is determined by the Lovain community detection algorithm [23].

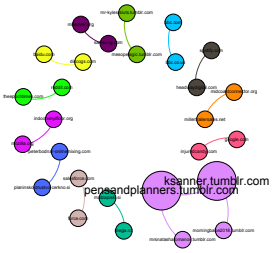


Figure 11: Confusion graph for the misclassifications in LOC1 that happen in more than one fold of the cross-validation and have different domain name length. We observe domains that belong to the same CDN (e.g., tumblr) or belong to the same entity (e.g., BBC, Salesforce). For others, however, the cause of the misclassification remains an open question.

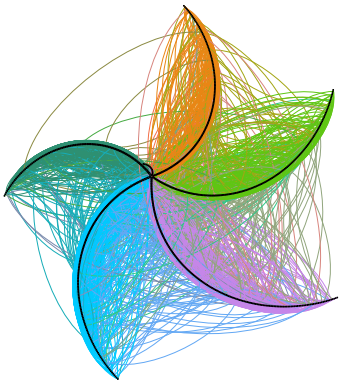


Figure 12: Confusion graph for all Tor misclassifications. We did not plot the labels to remove clutter. We observe that domains in one a "petal" of the graph tend to classify between each other.

A.5 Survivors and easy preys

Table 12: 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 13: 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 14: 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

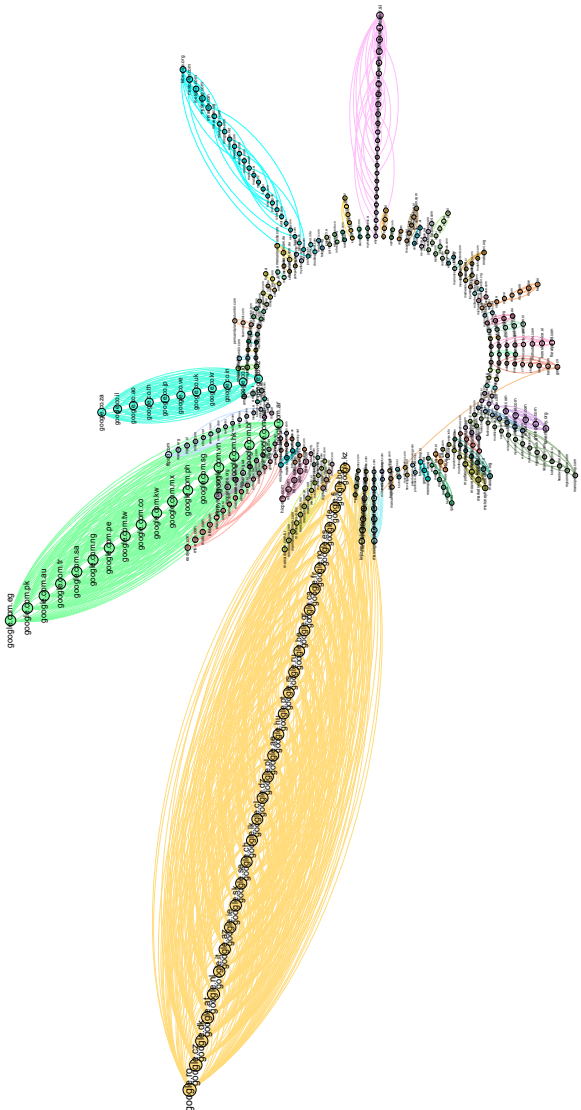


Figure 13: Confusion graph for all misclassifications in LOC1. We observe clusters of domains such as Google and clusters of domains that have the same name length. Interestingly, the only inter-cluster edge we observe is between one of the Google clusters and a cluster that mostly contains Chinese domains.

Bibliography

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

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

[3] Cherubin, G., Hayes, J., and Juarez, M. "Website fingerprinting defenses at the application layer". In *Proceedings on Privacy Enhancing Technologies (PoETS)* (2017), De Gruyter, pp. 168–185.

[4] Jansen, R., Juarez, M., Galvez, R., Elahi, T., and Diaz, C. Inside job: Applying traffic analysis to measure tor from within. In *Network & Distributed System Security Symposium (NDSS)* (2018), Internet Society.

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

[6] Juarez, M., Imani, M., Perry, M., Diaz, C., and Wright, M. Toward an efficient website fingerprinting defense. In *European Symposium on Research in Computer Security (ESORICS)* (2016), Springer, pp. 27–46.

[7] Juarez, M., and Torra, V. A Self-Adaptive Classification for the Dissociating Privacy Agent. In *PST2013, the eleventh annual conference on Privacy, Security and Trust* (2013), pp. 44–50.

[8] Juarez, M., and Torra, V. Towards a privacy agent for information retrieval. *International Journal of Intelligent Systems 28*, 6 (2013), 606–622.

[9] Juarez, M., and Torra, V. Dispa: An intelligent agent for private web search. In *Advanced Research in Data Privacy*, G. Navarro-Arribas and V. Torra, Eds., vol. 567 of *Studies in Computational Intelligence*. Springer International Publishing, 2015, pp. 389–405.

[10] Overdorf, R., Juarez, M., Acar, G., Greenstadt, R., and Diaz, C. How unique is your onion? an analysis of the fingerprintability of tor onion services. In *ACM Conference on Computer and Communications Security (CCS)* (2017), ACM, pp. 2021–2036.

[11] Rimmer, V., Preuveneers, D., Juarez, M., Van Goethem, T., and Joosen, W. Automated website fingerprinting through deep learning. In *Network & Distributed System Security Symposium (NDSS)* (2018), Internet Society.

[12] Siby, S., Juarez, M., Diaz, C., Vallina-Rodriguez, N., and Troncoso, C. Does encrypted DNS imply privacy? A traffic analysis perspective. *Submitted to the USENIX Security Symposium* (2020).

[13] Sirinam, P., Imani, M., Juarez, M., and Wright, M. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *ACM Conference on Computer and Communications Security (CCS)* (2018), ACM, pp. 1928–1943.

Curriculum

Marc graduated in Computer Engineering and Mathematics from Universitat Autònoma de Barcelona. He arrived to KU Leuven in February 2013 as an ERASMUS student and joined the PhD program at COSIC in the Autumn of the same year. Before starting his PhD, he was a research assistant and junior software engineer in the Research Institute on Artificial Intelligence of the Spanish Council for Scientific Research (IIIA-CSIC) in Bellaterra.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING
COSIC
Kasteelpark Arenberg 10, bus 2452
B-3001 Leuven
marc.juarez@kuleuven.be
https://securewww.esat.kuleuven.be/cosic/