Towards an Evaluation of a Recommended Tor Browser Configuration in Light of Website Fingerprinting Attacks

by

Fayzah, Alshammari

Thesis submitted

In partial fulfillment of the requirements

For the M.Sc. degree in

Computer Science

School of Electrical Engineering and Computer Science

Faculty of Engineering

University of Ottawa

Abstract

Website Fingerprinting (WF) attacks have become an area of concern for advocates of web Privacy Enhancing Technology (PET)s as they may allow a passive, local, eavesdropper to eventually identify the accessed web page, endangering the protection offered by those PETs. Recent studies have demonstrated the effectiveness of those attacks through a number of experiments. However, some researchers in academia and Tor community demonstrated that the assumptions of WF attacks studies greatly simplify the problem and don't reflect the evaluation of this vulnerability in practical scenarios. That leads to suspicion in the Tor community and among Tor Browser users about the efficacy of those attacks in real-world scenarios. In this thesis, we survey the literature of WF showing the research assumptions that have been made in the WF attacks against Tor. We then assess their practicality in real-world settings by evaluating their compliance to Tor Browser threat model, design requirements and to the Tor Project recommendations. Interestingly, we found one of the research assumptions related to the active content configuration in Tor Browser to be a reasonable assumption in all settings. Disabling or enabling the active content are both reasonable given the fact that the enabled configuration is the default of the Tor Browser, and the disabled one is the configuration recommended by Tor Project for users who require the highest possible security and anonymity. However, given the current published WF attacks, disabling the active content is advantageous for the attacker as it makes the classification task easier by reducing the level of a web page randomness. To evaluate Tor Browser security in our proposed more realistic threat model, we collect a sample of censored dynamic web pages with Tor Browser in the default setting, which enables active content such as Javascript, and in the recommended setting by the Tor Project which disables the active content. We use Panchenko Support Vector Machine (SVM) classifier to study the identifiability of this

sample of web pages. For pages that are very dynamic, we achieve a recognition rate of 42% when JavaScript is disabled, compared to 35% when turned on. Our results show that the recommended "more secure" setting for Tor Browser is actually more vulnerable to WF attacks than the default and non-recommended setting.

Acknowledgements

Profound thanks to Allah for his favor, grace, mercies and unrelenting love.

To King Abdullah bin Abdulaziz Alsaud, may Allah have mercy on him, for his exceptional role in promoting woman education and rights in Saudi Arabia. To King Saud University for sponsoring me during my studies.

To my supervisor, the incomparable Professor Carlisle Adams. I'm lucky and honored to have this opportunity to work closely under his supervision and with him. I'm grateful for every single moment I spent in his office discussing problems, solutions, struggles, and ideas. I owe a lot of my professional, academic and personal development to Carlisle Adams. His patience, endless support, encouragement, and guidance have helped me to cross a lot of boundaries. This thesis wouldn't have seen the light without his unlimited support and encouragement.

I would also like to mention the great opportunities I have had as one of his graduate students to work with industry and government software engineers, designers, and team leaders to solve real world problems. Such as our work with Canada Border Service Agency CBSA, Trend Micro, and the chance to volunteer in security related projects, such as the IBM Watson for Cyber Security project. That gave me the fascinating opportunity to witness and participate in the sharing and development of ideas which evolved into real world implementations, and experienced real world problems decomposed back into their constituent theoretical elements. All of this would not have been possible without Carlisle's support.

To the awesome people in Carlisle Adam's research group; David Bissessar, Maryam Hezaveh, Ali Noman, Xiaomei Zhang, Mike Wakim, Alain Tambay and Dr.David Knox;

I'm grateful for the wonderful and challenging times we went through together while solving problems and achieving milestones.

To Michael Mann from Wireshark, for taking the time to code for Tor Dissector, and leaving useful comments and suggestions for integrating Tor Dissector with Wireshark plug-ins. To Nick Mathewson from the Tor project for answering my questions about the baseline code for Tor. To Ian Goldberg's research group at the University of Waterloo, for having me in a workshop discussing website fingerprinting attacks. To Marc Juarez for gracefully sharing their datasets and code with us.

To all my friends, particularly Kenniy Olorunnimbe, Riyas Valiya, Dela De Yongester; thank you so much for always being there when I needed you.

To my brother "Rfytzy" Saad, who came with me to Canada and showed me how a person can play the role of a whole family; I love you and I will be always grateful for you. To my family, thank you for always loving me unconditionally.

To the "Anonymous" who doesn't want to be named; thank you so much for everything!

Dedication

In loving memory of Hammed Rahal Alshammari and Fayez Hammed Alshammari.

You are gone, but never forgotten. Father and Brother, I will always love you.

Contents

List of Tables

List of Figures

Acronyms

HMM Hidden Markov Model

HTML Hyper Text Markup Language

HTTP Hyper Text Transfer Protocol

HTTP Obfuscation HTTPOS

IP Internet Protocol

ISP Internet Service Provider

JAP JonDonym Anonymous Proxy

K-NN k Nearest Neighbor

ONI OpenNet Initiative

OP Onion Proxy

OR Onion Router

OSAD Optimal String Alignment Distance

PET Privacy Enhancing Technology

PII Personally Identifiable Information

RBF Radial Basis Function

RWB Reporters Without Borders

SSL Secure Socket Layer

SVM Support Vector Machine

TCP Transport Control Protocol

TLS Transport Layer Security

Tor The Onion Router

UNESCO United Nations Educational, Scientific and Cultural Organization

URL Uniform Resource Locator

VOIP Voice-over-IP

WF Website Fingerprinting

Part I

INTRODUCTION

Chapter 1

Introduction

"My computer was arrested before I was." Syrian Activist [51]

The Internet, specifically, the web, has become a major platform for freedom of opinion and expression [21] as it enables individuals who have access to it to seek, receive and impart opinions, ideas and information of all forms, instantly and inexpensively across the globe [21]. It was built fundamentally without any regard for protecting the privacy of its users. The absence of personal privacy protections built into the Internet and the constantly increasing threat of unlawful data tracking, censorship, and surveillance of web users' lives, force individuals around the world to seek and rely on privacy and anonymity technologies that can be used with the web to compensate for its lack of security and safety. Anonymity technologies have appeared as a potential good candidate to enhance people's human rights over the Internet so they could seek, receive and impart information without risk of disclosure, tracking and surveillance as mentioned in the United Nations Educational, Scientific and Cultural Organization (UNESCO) report [21] on the promotion and protection of the right to freedom of opinion and expression. The same report mentions The Onion Router (Tor) as a well-known anonymity tool. Tor

exists as one of the most widely adopted practical anonymous communications systems [52]. It is also a sophisticated PET that has the potential to be used as an anonymity, censorship circumvention and anti-surveillance tool [61].

Recent research shows that Tor browser is vulnerable to a traffic analysis attack that exposes the user's final destination, violating the expected privacy from Tor. 'Website fingerprinting' WF attacks refer to a form of traffic analysis attacks, where a local observer aims to infer(learn) the identity (Uniform Resource Locator (URL)) of a requested web page over an encrypted and/or anonymized web traffic. Here, the traffic content is generally assumed to be perfectly encrypted, so the attacker utilizes traffic meta-data such as packet timing, volume and direction, and pattern recognition techniques to extract fingerprints that uniquely identify the targeted web pages.

Before the end of 2011 [46], Tor was considered to be secure against this threat [29]. Since then, researchers have engaged in a war of escalation in developing website fingerprinting attacks against Tor; these studies showed that WF attacks are effective in Tor. However, some researchers in academia [33, 41] and Tor community [48] demonstrated that the assumptions of WF attacks studies greatly simplify the problem and don't reflect the evaluation of this vulnerability in practical scenarios. That leads to suspicion in the Tor community and among Tor browser users about the efficacy of those attacks in real-world scenarios.

In this thesis, we first survey the literature of website fingerprinting showing the research assumptions that have been made in the attacks against Tor. We then categorize them based on their practicality in real-world settings by evaluating their compliance to Tor Browser's threat model [50], its design requirements [50], and to the Tor Project recommendations [60] for using it.

Among the realistic assumptions is disabling the active content (JavaScript) while

browsing the web. It is reasonable to assume the client to disable her browser's JavaScript to protect herself from the privacy and security concerns related to the active content. This is also true in Tor Browser as it is the recommended setting by Tor Project [60]. However, surprisingly, enabling the active content (JavaScript) in Tor browser is reasonable too as it is configured by default to prevent web sites that rely on JavaScript from breaking while using Tor browser (i.e., for usability purposes only)[60]. Note that, there are several WF studies [41, 65, 15] that show the dynamic content of web pages complicates their WF attacks and lowers their accuracy rates. This implies that disabling JavaScript may be advantageous for the WF attacker, especially, in case of the dynamic web pages fingerprinting. This led us to our hypothesis which we discuss next.

1.1 Thesis Hypothesis

The active content configuration recommended by Tor Project may make Tor Browser more vulnerable to website fingerprinting attacks than the default configuration. In other words, the active content recommended setting (i.e., disabling JavaScript) in Tor Browser is advantageous for the WF attacker, especially in case of the dynamic web pages fingerprinting. That is because dynamic web pages vary in the size and number of objects they contain based on the active content (JavaScript) configuration in a browser. For example, when JavaScript is disabled, the embedded objects in the web page will not be requested. Hence, the number of requests sent by the browser and the total number of packets returned by the server will be smaller. Consequently, the amount of the randomized content included in the generated web traffic will be smaller too. This makes several traffic patterns generated from that web page more similar to each other and thus easier to identify than when JavaScript is enabled.

1.2 Thesis Contribution

- We survey the literature of WF attacks categorizing it into three domains: website fingerprinting attacks; countermeasures that have been proposed against such attacks; and research in the practicality of these attacks.

- We identify a conflict in the security and anonymity design requirements for the Tor browser. That is, disabling the active content is recommended to protect against Tor web browser fingerprinting and to protect from the privacy and security risks related to JavaScript, but in fact it makes website fingerprinting attacks more effective due to the reduced traffic randomness introduced by enabling active content such as JavaScript.

- We examine the effectiveness (in terms of classification accuracy) of our chosen classifier when the web pages are lightly dynamic and when they are heavily dynamic in order to determine the effect of dynamism on the success of website fingerprinting attacks.

- We implement a Traffic Analysis Framework in Python which we are making public to save other researchers time and engineering effort in this topic. Our framework integrates Tor browser with a traffic capturing tool called Dumpcap, to allow data collection in a real Tor network. It then extends the existing Dyer's Framework to parse Traffic, extract features and then use them for classification.

- We introduce Tor dissector, which we greatly re-factor as a Wireshark plug-in that can be useful for debugging purposes for Tor traffic. It also can be used for educational purposes for Tor protocol.

1.3 Thesis Outline

This thesis is organized into six (6) chapters. Chapter 2 introduces private web browsing over Tor. It starts with an overview of the web browsing describing the web page loading process and different types of the web pages. It then summarizes the privacy concerns related to web browsing in the public Internet. After that, it describes Tor components, protocol and dissector. More specifically, it describes the Tor architecture and the Tor design choices that are necessary to understand our analysis and experimental design. It then introduces Tor Browser as a PET to provide anonymous and private web browsing to conceal a user's Personally Identifiable Information (PII) at all the three levels. It also describes some of the relevant design choices to Tor Browser which led us to identify the recommended Tor configuration for JavaScript. In Chapter 3, we present the needed background related to the WF problem. We first describe the WF attacks threat model. Then, we describe the typical steps involved in website fingerprinting. In order to show where our research fits within the existing work in the WF field and how we ended up with our thesis hypothesis, we survey the existing work categorizing it into three domains: website fingerprinting attacks; countermeasures that have been proposed against such attacks; and research in the practicality of these attacks.

In Chapter 4, we describe our experimental setup to to test our hypotheses. We first describe the limitations that prevented us from using the prior relevant works' datasets in our analysis. We then describe the dataset we collected to avoid the aforementioned limitations. We also describe our list of web pages by explaining how we select, clean and categorize its web pages. After that, we explain how we use that list in our data collection process. We further continue describing the different software packages and libraries used in our data collection. We also explain how we configure each one of them. At the end of this Chapter, we describe how we implement our Data Generator script

to orchestrate the data collection process among the used software tools. Chapter 5 introduces the criteria adopted for the evaluation of our experiments. Analysis of the results is also presented in this chapter. It also discusses the limitations of our work. In the final Chapter 6, we present the concluding remarks, discuss our contributions, and then give a brief overview of our future work.

Part II

BACKGROUND AND LITERATURE REVIEW

Chapter 2

Private Web Browsing over Tor

The Web is fundamentally a network application (client/server) running over the TCP/IP protocol, which is among the fundamental building blocks of the Internet. The Internet was designed without any regard for preserving users' privacy or facilitating anonymity. As such, when people are surfing on the the web, their PII can be observed and collected from three different conceptual levels: TCP/IP level, Hyper Text Transfer Protocol (HTTP) level, and the application level [52].

This chapter starts with an overview of the web browsing describing the web page loading process and different types of the web pages. It then summarizes the privacy concerns related to web browsing in the public Internet. After that, it describes Tor components, protocol and dissector. More specifically, it describes the Tor architecture and the Tor design choices that are necessary to understand our analysis and experimental design. It then introduces Tor Browser as a PET to provide anonymous and private web browsing to conceal user's PII at all the three levels. It also describes some of the relevant design choices for Tor Browser which led us to identify the recommended Tor configuration for JavaScript.

Throughout this thesis, we use the networking terms "client" and "server" to describe

the two communicating end points involved in the web browsing transaction. We use "dynamic" and "active" content to refer to the embedded objects. We use "relay" and "node" interchangeably to mean a single Tor Onion Router (OR). Finally, "adversary" and "attacker" are used interchangeably to refer to any kind of malicious third party with certain capabilities who attempts to compromise a PET user's privacy.

2.1 Web Browsing

Web browsing is a web transaction that involves retrieving, rendering and displaying a requested web page from a remote web server. In the following, we explain the web page loading process and mention some of the optimization that affects its sequence. We then categorize web pages into static and dynamic. After that, we describe the web technologies that are used to build the dynamic web pages.

2.1.1 Web Page Loading

A web page is usually a document written in (Hyper Text Markup Language (HTML)) or comparable markup language, and can contain or embed various types of web resources such as style information which controls a page's look, scripts (active content) which add interactivity and/or dynamism to the page, and media such as images, sounds, and videos. Loading the web page involves loading the HTML file and all of the embedded resources. This happens through pairs of HTTP requests and responses. For more details about web page request flow, please see the survey done by Martinez *et al.* [52]. Figure 2.1 shows an example of a loading process of a web page that we created and hosted in a web server and retrieved through Tor Browser (described below in Subsection 2.4.1) after integrating it with Firebug [1]. Firebug is a Firefox plug-in that provides live debugging

and monitoring of any web page. The figure shows that the requested web page has 7 HTTP requests: One HTML, three CSSs, and three images. It shows also that they are all hosted in the same web server “”0x4eb.tech”. After a web page is retrieved, it is rendered and displayed by the browser. By default, a browser's rendering engine can display HTML documents and images. It can display other types of data via plug-ins or extensions, such as a PDF viewer or JavaScript plug-in [52].

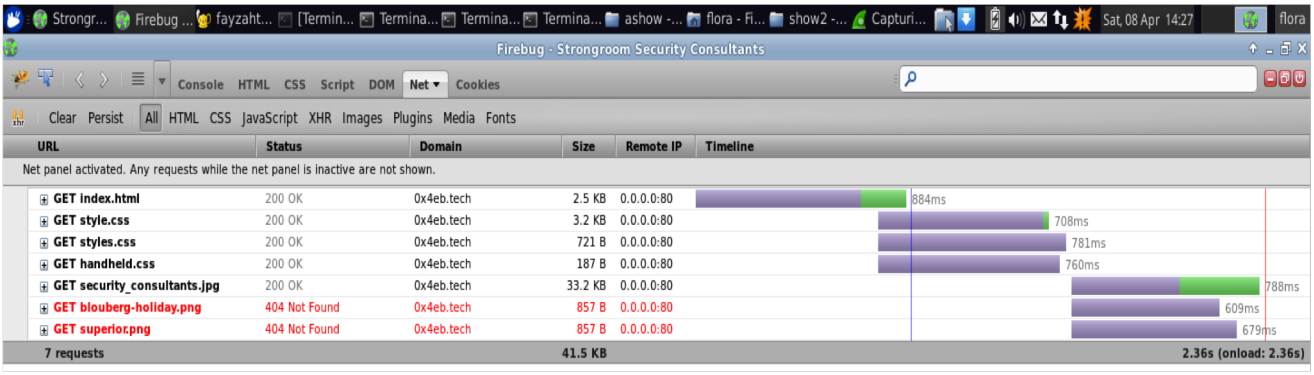

Figure 2.1: Web Page Request Flow Example

However, according to Cai *et al.* [12], browsing the web is not usually that simple because of the optimization that may be applied while loading web pages. For example, in the case of web pages that consist of multiple resources, the browser sends separate requests for each object by using a combination of multiple Transport Control Protocol (TCP) connections and pipelining in order to load the web page faster. Furthermore, the browser may pre-fetch the objects referenced in a web page before it has finished loading that page. Moreover, the sequence of HTTP requests and responses may vary each time the browser loads the page: some requests may be delayed because of packet re-ordering, and some requests (or responses) may be omitted if the browser caching is enabled at the time of the browsing or the active content is disabled in the browser. In the next Chapter 3 we show how web site fingerprinting attacks utilize the web page

loading process to extract features to identify the web pages despite the concealing of the HTTP requests and responses by encryption.

2.1.2 Web Pages Types

In general, web pages can be grouped into two categories according to the time at which the contents of the document are determined: Static and dynamic. Static web pages reside in a file that is associated with a web server and is delivered exactly as stored [17]. As a result each HTTP request for the same web page results in exactly the same response. On the other hand, dynamic web pages are created each time a browser requests a web page by running programs on the web server side, the client side, or both [17]. Thus, the content of a dynamic document can vary from one request to another and several visits to the same web page may generate different responses. It is worth mentioning though, that there is another case where several visits to the same web page may generate different responses too. That is when a web page (static or dynamic) content changes over time due to the content update. However, we are not trying to study these changes as there is an extensive research regarding the type and frequency of changes to such sites [16]. We are also not interested in studying their effect on the WF as there are already some studies regarding that (we explain this in more detail in our survey 3.3.3.2).

Based on the introduced definition of the dynamic web pages, we can, generally, say that a web page dynamism can be achieved by one or more of following two web technologies:

1. *Client-Side technologies* such as scripting, where the client's browser retrieves and runs the scripts (if its active content is enabled). The client-side scripting can be used to enhance the web page's interactivity and the user's experience [52]. As a

result, it can by controlled by disabling the client side active content. This prevents the embedded dynamic content from being requested, received and executed on the client's computer.

2. *Server-Side technologies* such as Microsoft's Active Server Pages (ASP), and PHP, where the server runs a program every time the web page is requested to decide exactly what to send the client based on what information was included in the request and what conditions are met at that time. After the page has been customized at the server-side, it will be retrieved, rendered and displayed at the client side. The Server-side scripting can be used to customize the displayed web page for the user; for example, the server sees the request coming from Canada so in the response it sends a Canadian-customized response (i.e., localization). Not as the client-side scripting, the server-side cannot be controlled from the client side.

In Subsection 4.2.3 we will show how we use and implement this categorization for our experimental purposes.

2.2 Web Browsing Privacy Concerns

Martinez *et al.* [52] showed that a web user's privacy can be compromised using information from the three different levels: TCP/IP level, HTTP level, and application level. In the TCP/IP layer, basically, the information that can be gathered to track web users is the Internet Protocol (IP) address and the port from which the user is making the request. The IP address also provides domain name, geolocation information, identifying Internet Service Provider (ISP), city, region, country and continent from which the request is being made. They also showed that the HTTP headers and HTTP cookies in the HTTP level can be used to collect PII about web users. These headers could reveal the

following information: the user's web browser (User-agent header), the URL of the Web site the user has visited previously (Referer header), and the user's mail address (From header). The cookie mechanism has been used broadly to track, profile and monitor user browsing activities [6]. Finally, they showed how the embedded objects (active content) in the application level (such as JavaScript, ActiveX) can represent a privacy threat since they can be used to fingerprint the user's machine [52]. In Subsection 2.4.2 we see how such concerns influence the active content configuration setting in Tor Browser.

2.3 Tor

Tor is a popular overlay network for anonymous communication over the Internet. It utilizes the onion routing that is based on layered encryption to hide the source of TCP traffic [19]. In this section, we first describe its components and how these components work together. Next, we describe Tor Protocol by explaining Tor traffic generated in circuit construction and then in navigation to a web page through Tor.

2.3.1 Tor Architecture

The fundamental architecture of the Tor network consists of three basic entities: **Onion Proxy (OP)**, **Onion Router (OR)**, and **Directory Servers**. An **OP** is software executed locally in the machine of each Tor user (client) [19]. It constructs circuits across the network and handles the requests of the user's applications and sends them through the established circuits [68]. The **OR**s are special proxies in charge of relaying the application data between the OP and the final destination (i.e., web server). Tor consists of around 7000 volunteer relays [62], which are routers that relay information for Tor clients. Finally, **directory servers** maintain and provide information such as

the list of onion routers available, the status of network topology, the OR public keys and the exit policies of each onion router [68]. Functions of onion proxy, onion router and directory server are all integrated into the same Tor software package which uses Tor configuration file (torrc) to configure a computer to have any combination of those functions [68]. Figure 2.2 shows the Tor network architecture. This figure is an edited version of a figure created originally by Bauer in [5]. Learning about Tor architecture is essential for understanding the WF attacks threat model we discuss in Subsection 3.1 and for understanding our analysis and observations overall in the thesis. The next paragraph explains how these components work together to send an application data through Tor network. We use Tor main research paper [19] as a reference for all of the sentences in the next paragraph unless otherwise mentioned.

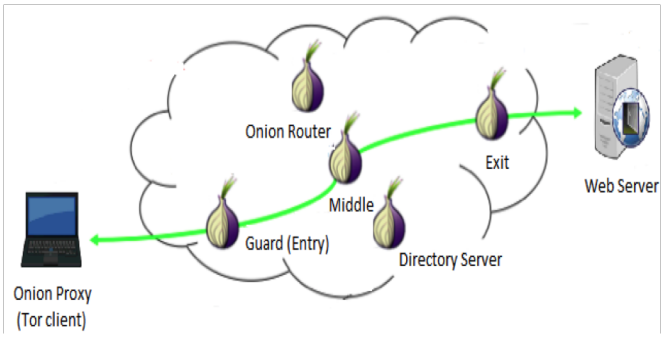

Figure 2.2: Tor Network Architecture

To send a TCP application data through Tor, the OP (Tor client) downloads a list of the available relays from the directory servers in the Tor network and chooses three relays, the guard relay, the middle relay and exit relay, respectively. To limit the rate of deanonymization, each client picks a guard from a list of three guard relays it will use for 30- 60 days, while the middle and exit relays are chosen randomly for each circuit [14]. It then constructs a path called a **circuit** of these three onion routers using incremental path-building and negotiating a symmetric key with each one of them. It uses that

circuit for about ten minutes before switching to a new circuit to limit the time a node on the path can observe the traffic (circuit rotations). Once the circuit has opened, Tor packs the application data into equal-sized structures of fixed size (512 bytes) named **cells**. It then encrypts each cell including the next node destination IP address, multiple times using the negotiated key with each one of the relays. After that, it sends the cells through the built virtual circuit using a number of multiplexed streams (each stream corresponds to a separate TCP connection at the exit relay). Each relay, then, decrypts a layer of the encryption to reveal only the next relay in the circuit in order to pass the remaining encrypted data on to it. After that, the exit relay decrypts the innermost layer of the encrypted data and sends the corresponding plain text (application original data) to its destination without revealing the source IP address of the client. For the recipient, the traffic between the exit and the destination server is not ciphered, which enable exit relay eavesdropping. Moreover, the traffic seems to have originated from the exit onion router, which makes Tor client retrieve different web pages for the same URL (web page localization) based on the location of the exist node [65]. In our data collection we keep Tor working as it is configured by Tor Project, which is a non-profit organization primarily responsible for developing and maintaining software for the Tor anonymity network. We only control the circuit rotations as we will explain in Chapter 4.

2.3.2 Tor Protocol

OP first sets up a TLS connection with the entry router OR1 by using the TLS protocol. Then tunneled through this connection, OP sends OR1 a CREATE cell and negotiates a key with OR1 by using the Diffie-Hellman (DH) key agreement protocol, where the parameter g is usually called a generator. OR1 responds with a CREATED cell, meaning a successful creation of a 1-hop circuit C1 between OP and OR1. To extend the circuit

one hop further, the OP sends OR1 a RELAY EXTEND cell, specifying the address of the middle router OR2. Upon receiving this cell, OR1 first decrypts the cell and negotiates secret keys with OR2 to create a second segment C2 of the 2-hop circuit, and then sends OP a RELAY EXTENDED cell encrypted by AES in the counter mode (AES-CTR). OP will decrypt the RELAY EXTENDED cell and use the information to create the corresponding keys with OR2. Similarly, OP can create a 3-hop circuit by sending OR2 a RELAY EXTEND cell through the established 2-hop circuit. When OR2 decrypts the cell, it discovers that the cell is meant to create another segment of the circuit to OR3. OR2 consequently negotiates with OR3 and sends a RELAY EXTENDED cell back to OP. OP will decrypt the RELAY EXTENDED cell and use the information to create the corresponding keys with OR3. Therefore, a 3-hop circuit is successfully constructed.

Wang *et al.* [68] explains how web browsing works over a Tor circuit. In this section we use their explanation and their Figure 2.3.

To access a web server (i.e., Bob), Alice's web browser will ask her OP, which is implemented as a SOCKS proxy locally, to establish a connection to Bob. When OP learns the destination IP address and port, it sends a RELAY BEGIN cell to the exit router OR3. The cell is encrypted as $\{\{\{\text{Begin(IP , Port)}\}k_3\}k_2\}k_1$, where each subscript refers to the key used for encryption of one layer of onion skin. When OR3 removes the last layer of onion skin by decryption, it recognizes the request of opening a TCP stream. Once a circuit has opened, the client communicates through the circuit using a number of streams. Each stream corresponds to a separate TCP connection at the exit relay, and streams are multiplexed in a circuit. A client may open many streams to load a single site. Using persistent HTTP connections, more than one resource from the same server can be downloaded through each stream to the port at the destination IP, which belongs to Bob. As a result, OR3 acts as a proxy of the client and establishes a TCP

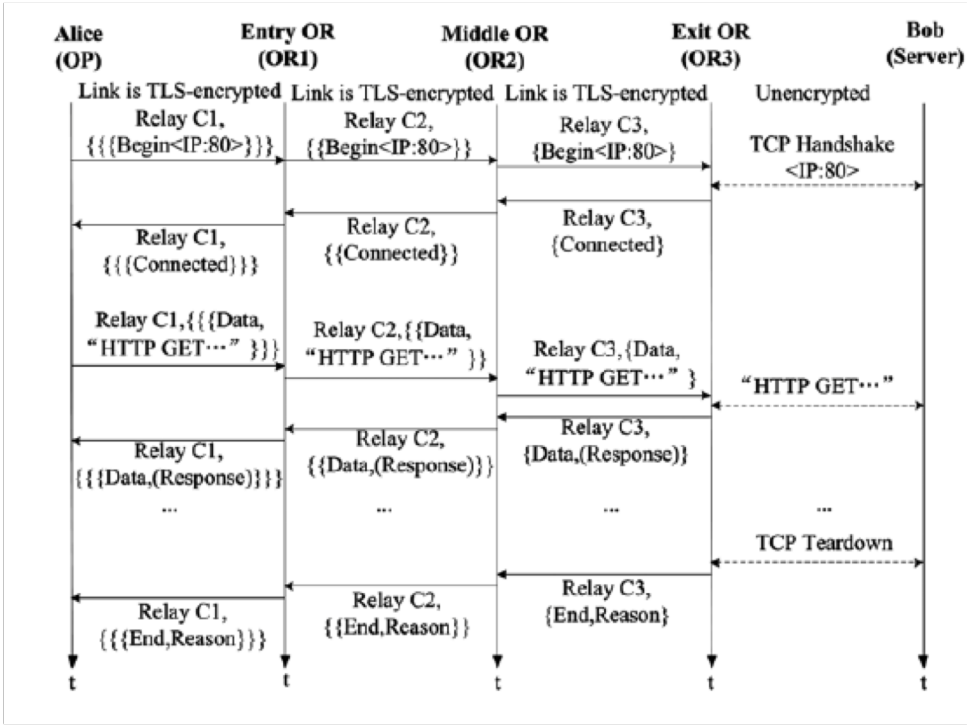

Figure 2.3: Tor Network Architecture [68]

connection to Bob by using the TCP handshake protocol. When the TCP connection to Bob is established, OR3 sends a RELAY CONNECTED cell back to Alice's OP. The OP then accepts data from Alice's web browser and sends a RELAY DATA cell to the exit router OR3, which is encrypted as $\{\{\{\text{Data, HTTP , GET . . . }\}k_3\}k_2\}k_1$. When OR3 recognizes the GET request initiated by OP, it asks Bob to return the requested webpage, and sends a RELAY DATA cell back to Alice's OP in response to OP web request. The whole process is transparent to Alice, although she needs to configure her web browser to use the OP. After delivering the requested data, the web server will close the TCP connection to OR3. OR3 then sends a RELAY END cell along the circuit to OP and sends nothing more along the circuit for that stream.

To help ourselves and the interested people to understand Tor protocol better, we

re-factored a 9-year-old Wireshark plug-in, which is explained next.

2.3.3 Tor Dissector

We patched Tor and Tor Browser to dump the encryption keys. However, this patch will not log the keys decrypted from the traffic of an Onion Router (i.e. a relay node on the Tor network), only the traffic coming to and from the Tor client on a specific machine (i.e. an Onion Proxy). Doing the former would require a different patch to Tor. Apart from being unethical, attempting this patch would potentially be illegal, as this could be considered as snooping on the traffic of other Tor users.

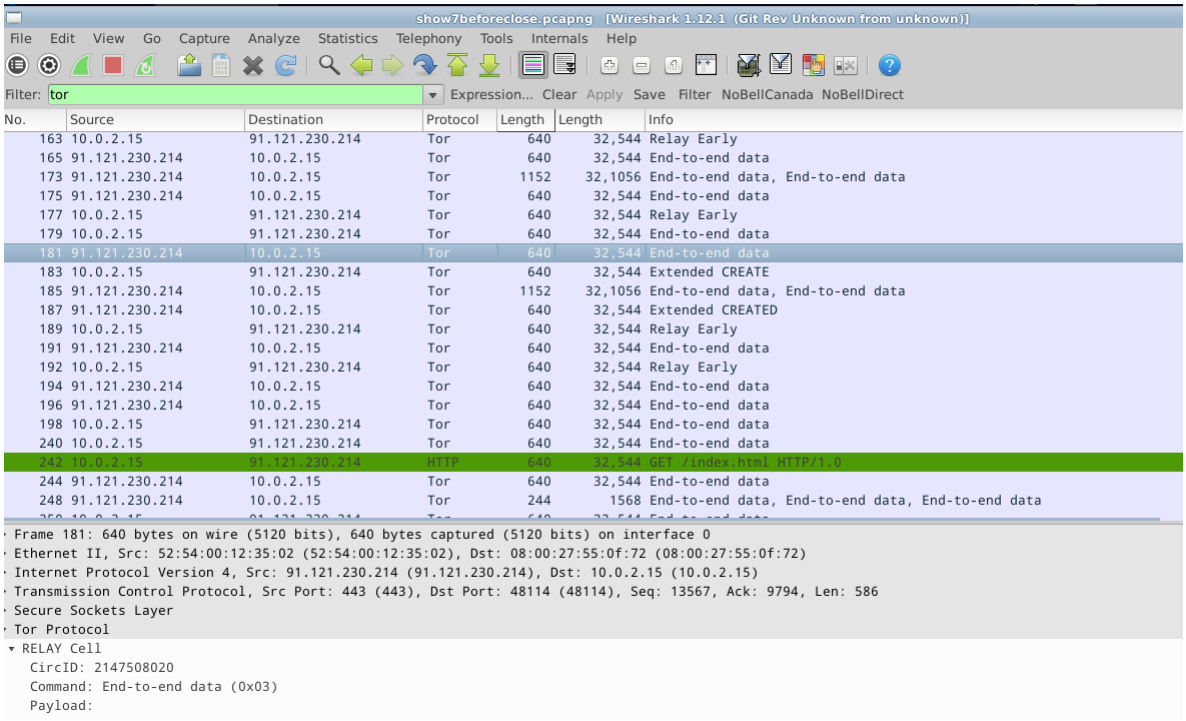

Figure 2.4: Tor Dissector

Modifications To Tor Browser Bundle We overrode the Tor Browser preferences to use our updated version of Tor instead of the default values bundled with Tor Browser.

Modifications To Tor A previous patch, originally designed by Steven Murdoch and implemented by Robert Hogan in 2009, had to be significantly modified to reflect the plethora of updates that have been applied to the Tor source code over the years. More specifically, we had to locate the specific unit of the application that logs the encryption keys, as the structure responsible for this function had considerably been changed. This patch logs the TLS master keys used to TLS-encrypt traffic between the client copy of Tor and other routers. It also logs the AES keys used to encrypt relay cells passed along the circuits created by this client. Wireshark then used these logs to decrypt the TLS and circuit streams it found in a traffic capture. We provide details of the modified files of the Tor source code in the next chapter.

Modifications To Wireshark We then feed the information that we logged to Wireshark. A new preference for the Secure Socket Layer (SSL) protocol was implemented in Wireshark using User Accsess Table (UAT). This is a data structure defined to maintain a dynamically allocated table accessible to the user via a file and/or GUI interface. We also implemented a Tor dissector as a plug-in for Wireshark.

2.4 Web Browsing over Tor

There are currently two options to browse the web over Tor. The first option is called **Torified Browser** [57], where the user can take any browser and configure it manually to direct web traffic over Tor. The second option is by using **Tor Browser** [59] which is a special version of Firefox browser that has been patched and pre-configured by Tor Project to satisfy a set of security and privacy requirements [50]. The former option is not recommended by Tor Project, as a number of privacy leaks may happen in the HTTP level and the application level as we explained above in Section 2.2, which can

then violate the anonymity provided by Tor. The latter, is the latest recommended way to use Tor for web browsing as it provides an easily-deployable solution to protect against almost all the privacy concerns presented in the three layers above described in [52]. Thus, the user has a comprehensive solution for her anonymous and private web browser.

Next, we describe briefly Tor Browser first and then its design decisions that are related to the active content.

2.4.1 Tor Browser

Tor Browser launches Tor and manages it through the Tor Launcher add-on, which provides the initial Tor configuration splash screen and bootstrap progress bar. In our data collection we disable this screen to facilitate our web browsing simulations. We explain further how we accomplish this in Chapter 4. We also replaced the bundled Tor with our patched version of Tor described in Subsection 2.3.3. Tor Browser also includes HTTPOS (HTTP Obfuscation)-Everywhere plug-in to help protect against potential Tor exit node eavesdroppers as we mentioned in Subsection 2.3.1. To provide end users with optional defense-in-depth against JavaScript, Tor browser also shipped with NoScript plug-in. We investigate this further in the next Subsection.

2.4.2 The Active Content Setting in Tor Browser

The active content configuration in the Tor Browser is controlled by the NoScript [39]. NoScript is a Firefox plug-in that controls the active content in a web page such as JavaScreipt. It enables JavaScript by default in Tor Browser for usability purposes. That is, some websites will not work with JavaScript disabled and people may give up on Tor as they dont know what JavaScript is or how they can configure it in the browser [60].

However, for more security and anonymity, Tor Project recommend disabling JavaScript completely to protect against security and web privacy concerns such as the ones we describe in Section 2.2. Based on this observation we defined two settings for Tor Browser in regard to the active content:

1. **The Recommended Setting** (JavaScript is off): Tor Browser that is re-configured by the user to disable JavaScript as recommended by Tor Project [60].

2. **The Default Setting** (JavaScript is on): Tor Browser as shipped by Tor Project [50].

In our experimental design in Chapter 4, we further show how we implement each one of these settings.

2.5 Summary

In this chapter, we started first by describing the web page loading process. We showed it as a sequence of exchanged requests and responses between a client and a web server. We pointed out that it could also be a complicated process that doesn't necessarily follow the exact basic HTTP protocol due to several optimization factors such as pipelining. We then classified the web pages as static and dynamic and defined the web technologies that are used to make web pages dynamic in the client and server side. We also showed that while web browsing, the PII can leak from three conceptual levels: the TCP/IP level, HTTP and the application level (the embedded web resources in the web page). Next, we described Tor network components and how they work together. We also described in detail Tor Protocol that is related to Tor Traffic generated from Tor circuit construction, and web page loading over Tor. We used our re-factored Tor dissector to help explain that part of Tor Protocol. After that, we introduced Tor Browser as a popular PET that

is designed to conceal user's privacy at all three levels by using Tor to anonymize web traffic and by applying several patches and configurations on a special version of Firefox browser to protect the user's privacy while browsing the web.

In the next chapter, we give the needed background about website fingerprinting attacks. We describe its threat model and procedure. We also show how they utilize the web page loading process patterns to identify the user's final destination (i.e., the requested web server) despite her use of the web PETs by surveying the literature of website fingerprinting attacks. We then extend this survey to cover the proposed defenses to counter those attacks and the practicality of the research assumptions. We then conclude with our hypothesis that we formulate from the survey.

Chapter 3

Website Fingerprinting

In the previous chapter, we presented the needed background related to the private web browsing where we presented Tor Browser as a popular PET to protect privacy while browsing the web. However, recent research shows that Tor and consequently Tor browser, like other web PETs, are vulnerable to an attack called Website Fingerprinting. The website fingerprinting attack refers to a form of traffic analysis attacks, where a local observer aims to learn the identity (i.e., URL) of a requested web page over an encrypted and/or anonymized web traffic. Here, the traffic content is generally assumed to be perfectly encrypted, so the attacker utilizes traffic meta-data (such as packet timing, volume and direction) and pattern recognition techniques to extract fingerprints that uniquely identify the targeted web pages, thus violating the expected privacy from the used PET.

In this chapter, we give the needed background for the WF problem. We first describe the WF attacks threat model. Then, we describe the typical steps involved in website fingerprinting. In order to show where our research fits within the existing work in the WF field and how we ended up with our thesis hypothesis, we survey the existing work categorizing it into three domains: website fingerprinting attacks; countermeasures that

have been proposed against such attacks; and research in the practicality of these attacks.

3.1 Website Fingerprinting Threat Model

Website fingerprinting attacks assume a user wants to hide which web pages she requests from the web against third parties [29]. This can be achieved with browsers that utilize web PETs. Usually, the user needs to install a dedicated client software that establishes an encrypted link (i.e., tunnel) to at least one proxy on the Internet. This proxy then relays the HTTP requests of one or multiple clients to the various destination web servers.

The attacker in a WF scenario is generally a curious eavesdropper who aims to identify which web page a specific targeted client is loading. WF attacks further assume that the attacker is able to record the traffic of the victim. For example, he could have wiretapped the connection between the victim and the PET system, so that he can retrieve the victim's true IP address from the traffic, and is able to identify the victim based on her IP address. Figure 3.1 depicts this general scenario; it shows the respective locations of the attacker and the proxy. This figure is a modified version of a figure given initially by Dyer *et al.* [22]. In case of Tor, the attacker locates himself between the client and the guard node shown in Figure 2.2.

Some examples of such attackers who have that sort of power according to the attacker classification in [47] includes local administrators, the victim's ISP, government and secret services. This also includes any other eavesdropper that can monitor the link between the victim and the used PET. Moreover, the attacker is not allowed to 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 [46]. To sum up, website fingerprinting methods may enable a passive, local observer (eavesdropper, according to their definitions in [18]) to circumvent the security provided by web PETs.

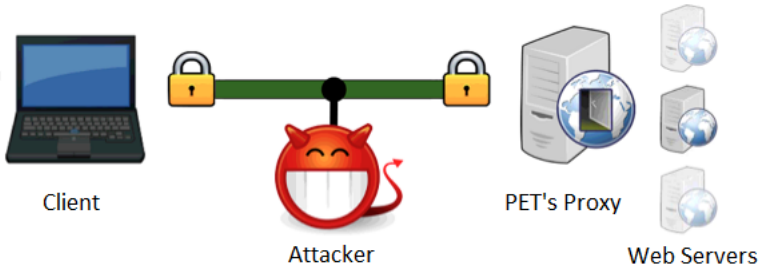

Figure 3.1: Website Fingerprinting Attacks Scenario

3.2 Website Fingerprinting Procedure

Website fingerprinting is commonly formulated as a data mining classification problem. In practise, the attacker first identifies n of the web pages he likes to identify, such as a set of censored web pages. He then collects many examples of traffic dumps from each of these web pages, which we call **training data** [29]. He does so by performing web-requests through the PET he assumes the victim uses as well, while recording the transferred packets in packet capture files (i.e., pcap files). A pcap file is the raw format of a traffic dump generated from visiting a specific web page. It provides information about the IP layer packets, e.g., the length of the packet, the time the packet was sent or received, the order in which the packets were sent and received. The attacker then pre-processes the pcap files by converting them into traces [46]. A trace is a parsed pcap file in such away that the lengths, directions and timings are extracted for each ciphertext in the pcap file. The attacker then uses these traces to extract web pages fingerprints. The web page fingerprint is the set of the identifiable features that can be extracted from several traces of the web page. The attacker then convert these features into traffic instances. A traffic instance is the formatted fingerprint to be used as training and testing examples in the classification phase. The attacker then uses these instances to train a machine learning algorithm (usually a classifier) to be able classify future

instances of a specific web page. Later, wiretapping on the victim's traffic, the attacker similarly collects data which we call **test data** [29]. He then passes it into his classifier and checks if the client visited one of the n web pages. In Chapter 4, we simulate this procedure as a part of our research methodology.

Figure 3.2 depicts this procedure as four phases: data collection which generates the pcap files; traffic pre-processing which generates the traces; features extraction which generates features that converted to instances to be consumed by the last phase: the classification.

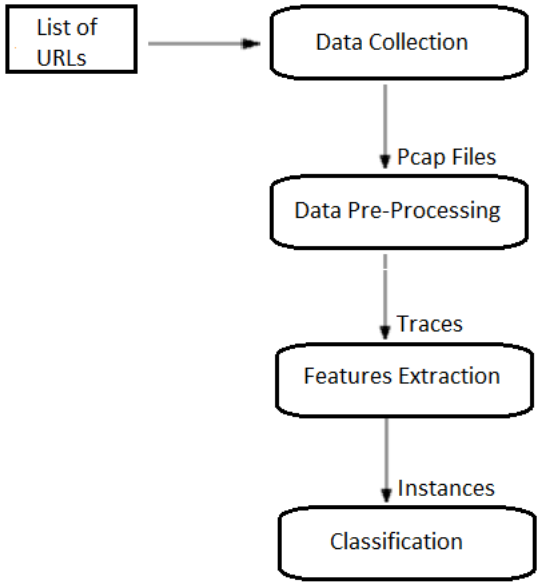

Figure 3.2: Website Fingerprinting Procedure

3.3 Website Fingerprinting Survey

Back in 1997, Wagner and Schneier pointed out that traffic analysis can pose a subtle threat in the SSL-encrypted web traffic [63]. They relayed an observation of Yee that

examination of cipher text lengths can reveal information about URL requests such as length of the requested URL, and the length of the HTML data returned by the Web server. Since then, traffic analysis vulnerability on web traffic has become an area of concern for advocates of web PETs as it can allow an eavesdropper to eventually identify the accessed Web page, endangering the protection offered by those PETs. Consequently, several studies have been conducted to evaluate that vulnerability in different PETs, with a sub-field known as "website fingerprinting attacks" having appeared within the field of the traffic analysis attacks. These areas of research have ranged from studies to show the feasibility of these attacks and to assess their severity in different PETs, to studies that try to improve the practicality of those attacks.

In this section, we survey the literature of WF attacks, categorizing it into three domains: website fingerprinting attacks; countermeasures that have been proposed against such attacks; and research in the practicality of these attacks. We further divide the attacks as in [29] into file-based attacks and packet-based attacks. We also divide the defenses as in [11] into network-level defenses and application level defenses. We finally divide the assumptions into reasonable assumptions and unreasonable assumptions. Figure 3.3 shows the overall picture of our survey. More details are given about those categorization in the following sections.

Note that there are other related works which we are not going to survey. These include studies that concentrate on the detection of other distinct characteristics of network traffic instead of website fingerprinting attacks, such as the language of a Voice-over-IP (VOIP) call [9] and spoken phrases in encrypted VOIP calls [10]. We also, as Wang *et al.* [66], don't consider other active attacks that can accomplish the same goal of website fingerprinting attacks such as packet spoofing [26], and measuring memory fingerprints [31]. We also don't consider website fingerprinting attacks in Tor hidden services [35].

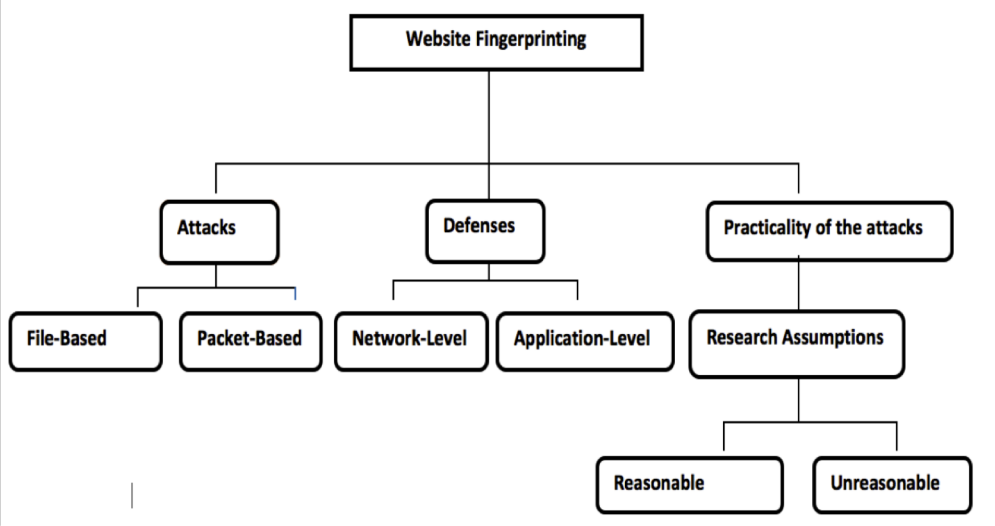

Figure 3.3: Website Fingerprinting Survey

3.3.1 Website Fingerprinting Attacks

Since 2002, several previous works have demonstrated the effectiveness of WF attacks against different web PETs. In this Subsection,we survey the proposed attacks until 2016 showing how they evolve from a potential vulnerability in the SSL protocol to an attack in Tor. We also show the different web page fingerprinting techniques that have been used in the literature to utilize the web page loading process. We, additionally, describe how the WF attacks evolved to work with the newer versions of HTTP and with the different web PETs. We finally, highlight Panchenko's SVM more as it is the classifier we pick to use in our work.

These WF attacks can be categorized as in Herrmann [29], based on the source of the fingerprints (i.e., where the distinguishable features are extracted from), into file-based attacks and packet-based attacks.

3.3.1.1 File-Based Attacks

File-based attacks work by utilizing the sizes of all HTML files and all objects referenced therein to identify which specific webpage was requested over a simple PET that merely encrypts the content of the web traffic, such as SSL [29]. HTTP /1.0 was the dominant protocol deployed in the web, it clearly specified that each web object should be transmitted through a new TCP connection. As a result the number and sizes of objects were easily obtained by examining TCP connections. Both features were good enough to distinguish websites. Cheng and Avnur [15] described the first implementation of the attack to identify which specific URL was accessed on a known server over an SSL-protected connection. Hintz [30] then introduced the idea of WF attack to disclose the security risks when the server is not known (e.g. SafeWeb proxy). In the same year 2002, Sun *et al.*[54] also presented a similar attack against the SSL encrypted connection. They proposed Jaccard's coefficient [28] as a metric to measure similarity between observed and pre-collected traffic patterns and achieved the detection rate of about 75%.

These early works showed the general feasibility of the website fingerprinting attack by considering the total sizes of resources. However, they assumed that each request is associated with a separate TCP connection, a constraint that only holds for early versions of the HTTP protocol. Nowadays, HTTP makes use of persistent connections and pipelining (cf. RFC 2616, section 8.1 [25]) to improve performance. There is no need to open a new TCP connection for each web object. Hence, it is no longer possible to trivially distinguish between single web object requests. Furthermore, the emergence of practical anonymous systems such as Tor completely removed the information about objects as they established an encrypted tunnel and mapped TCP connections to different logical channels [29]. Thus, the aforementioned attacks were no longer feasible and the next category of attacks (packet-based attacks) has been proposed in the literature.

3.3.1.2 Packet-Based Attacks

Bissias *et al.* [8] were the first to utilize some fine-grained features at the IP packet level instead of web object sizes to create fingerprints. They apply the "rather crude metric" of the correlation coefficient to the packet size and packet inter-arrival time observed during the download of a website over OpenSSH tunnel. To further improve that attack, Liberatore and Levine [36] applied Naive Bayes classifier to achieve 70% detection rate. In 2009, Herrmann *et al.* [29] were the first to apply website fingerprinting to the anonymization networks [32] [7] and Tor [19] as well as on OpenSSH, OpenVPN, Stunnel, and Cisco IPsec-VPN. They extracted the direction, packet sizes and frequency to represent fingerprints, and used the Multinomial Naive Bayes classifier to identify them. They achieved recognition rates above 90% for single-hop systems, but only 20% for JonDonym Anonymous Proxy (JAP) [32], and as low as 2.95% for Tor. Their results indicate that Tor is more resistant to WF attacks than the other PETs they used and more challenging to the attack. Lu *et al.* [37] found that website fingerprinting can be improved by considering information about packet ordering. They utilized the Edit Distance [37] to measure similarity between the incoming and outgoing packet sequences to reach 81% identification accuracy.

In 2011, Panchenko *et al.* [46] significantly improved the identification accuracy from 2.95% to 54% in Tor. Their results spawned a significant amount of interest in the research community and Tor community. As we are using their classifier in our thesis (Chapter 4 and Chapter 5), we are describing it separately below. After that, Cai *et al.* [13] replaced the Radial Basis Function in SVM [46] with the Damerau-Levebshtein Distance, which made a great success with over 80% accuracy. They utilized an SVM, but their features were based on the Optimal String Alignment Distance (OSAD) [65] of packet sequences. They were the first to study the recognition of different pages of

a website and the effect of clicking on embedded links, i.e., browsing within the same website, using the Hidden Markov Model (HMM). Though such a user behavior turned out to be detectable with a high probability, their study was limited to two websites only.

To improve the accuracy further, Wang *et al.* [65] improved the optimal string alignment distance approach of Cai *et al.* [13] and enhanced the data preparation by using Tor cells rather than TCP packets and provided an experimental method to remove Tor SENDME cells. With these improvements they obtained recognition rates of better than 90 % for both the closed-world (100 URLs) and the open-world (1,000 URLs) scenarios. In their subsequent work, the authors further improved the recognition rates in larger open-world scenarios (bigger than 5,000 URLs) using a novel k Nearest Neighbor (K-NN) classifier, which also significantly reduces the time necessary for training, compared to previous results [64].

Panchenko Panchenko *et al.* [46] Utilized SVM, which is a supervised learning method known for its high classification accuracy. The key idea is the representation of instances as vectors in a vector space. In the case of WF, the features and raw data are derived for one page traffic and represented as a vector. Based on training data, the classifier, then, tries to fit a hyperplane into the vector space which separates the instances (web pages traffic) that belong to different classes (i.e., URLs). The plane is fitted such that the accumulated distance between the closest instances (support vectors) and the plane is as high as possible to ensure a clear distinction between the classes. In cases where the vectors are not linearly separable, the vector space is transformed into a higher dimensional space by using kernel trick.

We further describe how we use their classifier in our work in Section 4.6 and how we apply it in our own dataset in Section 5.1.

3.3.2 Website Fingerprinting Defenses

While WF attacks emerged as a serious threat to users' privacy, people began to research defense mechanisms. Several countermeasures have been proposed to protect against website fingerprinting WF attacks. In this subsection, we briefly survey the defences until 2016 showing the different proposed defenses that target reducing the amount of the traffic information which is used in the fingerprinting attacks.For more information about the defenses, please refer to Juarez *et al.* and Dyer *et al.* [34, 22], respectively. We also show that they generally have high cost in terms of added delay and bandwidth overhead, which make them impractical for deployment in real-world systems.

The existing defenses in the literature can be divided into two categories as in Wang *et al.* [11] based on where the defense mechanism takes place: network-level defenses and application level defenses.

3.3.2.1 Network-Level Defenses

Network-level website fingerprinting defenses aims to reduce the amount of information exploitable by the attacker by operating on a per-packet level such as padding packets, splitting packets into multiple packets (fragmentation), or inserting dummy packets [11]. Some of them may also introduce additional traffic such as [46]. Here are some examples from the existing works.

Padding as a basic countermeasure was first introduced by Liberatore and Levine [36]. Wright *et al.* [71] proposed traffic morphing which aims to adapt a complete packet trace such that it looks similar to another packet trace. Lu, *et al.* [37] extended traffic morphing to operate on n-grams of packet sizes; it pads and fragments packets so that n-grams of packet sizes match a target distribution. However, Dyer *et al.* [22] showed traffic morphing and padding schemes to be ineffective as a defense against the WF

attacks they evaluated.

There are also a number of countermeasures that work by adding a continuous data flow. Panchenko *et al.* [46] proposed creating background noise by loading a random website in parallel with the actually requested website thus obfuscating the real traffic. However, Wang *et al.* [64] pointed out that this defense is not powerful enough to prevent website fingerprinting attacks if the traffic overhead is to be kept reasonable. Dyer, *et al.* [22] introduced BuFLO (Buffered Fixed- Length Obfuscation), which pads or fragments all packets to a fixed size and sends packets at fixed intervals, injecting dummy packets when necessary. However, Cai *et al.* [13] stated that BuFLO may reveal the total transmission size under certain conditions. Furthermore, it introduces a high overhead in bandwidth and time. It is also not able to adapt for congestion. To overcome these flaws, the authors proposed Congestion-Sensitive BuFLO (CS-BuFLO). For even further performance improvement of BuFLO, Cai *et al.* [12] proposed Tamaraw which is a heavily modified version of BuFLO to treat incoming and outgoing packets differently.

Glove [43] is a defense that utilizes knowledge of website traces for traffic morphing in a way that the attacker can only identify the cluster to which the web page belongs, but not the web page itself. The cluster is a large group of similar web pages with an added small amount of cover traffic to make all the pages within the cluster indistinguishable. A similar defense- called Supersequence- is proposed by Wang *et al.* [64]. However, to be successful, these defenses need to have a-priori information about each page to be protected.

To overcome this, Wang and Goldberg [67] proposed the Walkie Talkie defense that enables the Tor Browser to transmit in half-duplex mode. It buffers packets in one direction and sends them in bursts together with dummy traffic. This usually leads to lower bandwidth overhead compared to Tamaraw or Supersequence and allows for

a fixable packet rate to deal with congestion. Finally, Cai *et al.* [12] analyzed WF attacks and defenses from a theoretical perspective using a feature-based comparative methodology to learn to which extent certain defenses hide which features.

3.3.2.2 Application-Level Defenses

Application-level website fingerprinting defenses aim to reduce the amount of information exploitable by the attacker by operating on the application level (that is, the HTTP protocol level). Unlike the network level defenses, they don't introduce additional traffic. Instead, they alter the sequence of HTTP requests and responses to further obfuscate the user's activity. The following are some examples from the existing works.

Luo *et al.* [38] proposed HTTP Obfuscation which alters packet sizes, web object sizes, and timing by modifying HTTP and TCP requests so the client can, for example, change the traffic pattern of requesting a large resource to the traffic pattern of multiple requests of small resources. As a response to the evaluation of Panchenko *et al.* [46], the Tor Project released an experimental patch of the Tor Browser Bundle that implements a defense called randomized pipelining [49], that randomizes the quantity of requests processed in parallel (i.e., the browser pipeline size) and the order of requests for embedded website objects. However, both Cai *et al.* [13] and Wang *et al.* [65] showed that these defenses are not as effective as assumed and, in the case of randomized pipelining, might even lead to increased recognition rates on small datasets.

3.3.3 Practicality of Website Fingerprinting Attacks

Unfortunately despite all of the existing defense schemes which are described in Subsection 3.3.2, recent work conducted by Juarez*et al.* [34] demonstrated that none of those defenses are practical for deployment in real-world systems because of their high cost

in terms of added delay and bandwidth overhead. Moreover, these defenses have been designed to counter attacks that have been criticized for assuming unrealistic attack conditions in the evaluation setting. Some developers in Tor Browser [48] and academia [33, 41] have argued that the high success rates reported by the attacks are based on research assumptions that are unrealistically advantageous for the adversary. And thus, the reported effectiveness is not indicative of the practical effectiveness one can expect for actual Tor users.

In this subsection, we survey the literature of website fingerprinting showing the research assumptions that have been made in the attacks we describe in Subsection 3.3.1. We then categorize them based on their practicality in real-world settings by evaluating their compliance to the Tor Browser threat model and design requirements as described in the Tor browser documentation [50] and to the Tor Project recommendations and guidelines when using Tor Browser as given in Tor Project's website [60]. We also discuss the relevant previous studies for each one of the assumptions to support and justify our categorization. We then conclude with the set of the survey findings that seeds our thesis hypothesis in Subparagraph 3.3.3.2 .

3.3.3.1 Research Assumptions

The WF attacks research assumptions have been listed in two studies [41, 33]. The first work [41] is in 2012, which compiled the research assumptions in the WF attacks against several PETs generally, not only the ones evaluated in Tor. They listed eight assumptions and divided them into explicit and implicit assumptions. According to their classification, explicit assumptions refer to the assumptions that are related to the web pages, the configuration of the software that is used to load those web pages, and the assumptions that is related to the web pages traffic parsing. On the other hand, the

implicit assumptions are those assumptions that do not reflect the user diversity such as evaluating the WF attacks in the same operating system, browser, plugins and network. Please see Miller's work in [41] for a summary of their research assumptions along with the research papers that make those assumptions.

The second work is by Juarez *et al.* [33] in 2014, where the authors were more specific in considering the research assumptions in the existing attacks against Tor only. They provided a list of six research assumptions after they divided them into assumptions related to the client-settings, attacker, and those related to the web. Please see [33] for their list of assumptions and their summary about the studies that make those assumptions explicitly.

We found both lists have the same assumptions except two assumptions that are related to the web page dynamism: The active content assumption mentioned in [41] and the localization assumption mentioned in [33]. It is worth mentioning also these two lists in addition to our list are all similar to the list provided by Herrmann *et al.* [29]. However, our list is the only list that focuses on Tor Browser specifically not on all PETs, or on Tor generally. This helps us to give a more accurate practicality evaluation of our list of assumptions. We compile this list from the assumptions which all at once explicitly or implicitly in the WF attacks we surveyed in Subsection 3.3.1. We ended up with the following list:

1. The attacker knows the victim is using the Tor browser and may use it himself to create a database of website fingerprints.

2. The attacker is able to configure and use a similar Internet connection like the victim.

3. The attacker knows the victim's browser configuration.

4. The attacker can extract Tor browser web traffic from the victim's traffic, i.e. the attacker is able to filter all background network traffic produced by other applications or other connections going through the same Tor circuit.
5. The attacker knows which sequence of packets corresponds to an individual web page.
6. The victim's browser is configured to not query for software updates.
7. The victim's browser has caching disabled.
8. The visited web pages are all known to the attacker in advance.
9. The visited web pages are the home pages of different websites.
10. The visited web pages update their content rarely.
11. The victim is always going to visit the localized version of the websites.
12. The victim configured her browser to disable JavaScript.

3.3.3.2 Assumptions Evaluation

In the following, we categorize the assumptions in our list based on their practicality in real-world settings by evaluating their compliance to Tor Browser threat model and design requirements as described in the Tor browser documentation [50] and to the Tor Project recommendations and guidelines when using Tor Browser as given in Tor Project's website [60]. We also discuss the relevant previous studies that most relevant for each one of the assumptions to support and justify our categorization. Our categorization includes: unreasonable assumptions and reasonable assumptions.

Unreasonable Assumptions We consider the assumptions from 4 until 11 to be unreasonable. We justify that as follows:

The attacker can extract traffic corresponding to each individual web page (assumptions 4, 5 and 6): Assuming the attacker to be able to isolate and identify traffic generated from a specific web page load results in an unrealistically high accuracy rate, if an attacker is unable to actually isolate the traffic in practice. Juarez *et al.* [33] empirically showed the identification accuracy drops dramatically if a user performs multi-tab browsing. Above all, real-world studies found that users tend to have multiple open tabs or windows [33], which allow them to load several pages at once. This may be particularly applicable to Tor users due to its known performance issues [3]. Another recent implemented feature in Tor browser that makes this assumption unrealistic is the auto-update functionality [50], where the browser checks automatically for updates and downloads them in the background. Although Tor browser allows the user to disable this or to opt-in for manual updates, it is unreasonable to assume that a Tor browser user would turn this feature off as this would be against the general software security practises. Furthermore, previous work has shown that it may be difficult to extract individual web sessions from network traces in reality [33]. This difficulty increases when the user uses Tor for other applications other than web browsing such as Tor-enabled XMPP, IRC client, or even sends her complete operating system traffic over the Tor network [48].

The victim's browser caching is disabled (assumption 7): Assuming the browser cache (disk and memory caching) to be completely disabled is an unrealistic advantage for the attacker, as turning caching off makes the traffic pattern generated for a webpage more similar than when caching is on. That in turn makes the attacker's task much simpler as he doesn't need to take into consideration the different possible

versions of a web page when the caching is enabled. Miller *et al.* [41] conclude that enabled-caching has a high potential to frustrate existing website fingerprinting attacks as they noticed the total amount of traffic decreases, making the traffic pattern generated for a webpage vary with caching enabled. Moreover, Perry [48] pointed out each web page won't have only traffic patterns consisting of the cached vs non-cached; rather, it would have more patterns due to the all possible arbitrary combinations of the web page content elements that may be cached by the browser from a previous visit.

Although all the existing website fingerprinting studies that we surveyed evaluated their attacks while the caching was disabled (as was the case in [46] and [65]), two of them did a "preliminary" evaluation of the effect of caching on their classifier's efficiency. The first study was conducted by Herrmann *et al.* [29] where they conclude that caching affects their attack accuracy only moderately but they emphasize that this effect results from the deterministic nature of the page retrieval process used in their setup which may not accurately reflect the diversity of possible cache states in reality. Miller *et al.* [41] has criticized their study because of the use of several other assumptions related to the caching which unrealistically increased the traffic vulnerability in their study. One such assumption was the increase of the browser cache size from the default 50MB to 2GB, thereby preventing cache evictions. The second study carried by Cai *et al.* [13] has also done some work for the case when the client has cached the destination web page.

Moreover,Tor browser's memory caching is enabled by default and there is no reason to believe that a user would disable it as caching speeds up the web page retrieval process. This may be particularly applicable to Tor Browser users due to Tor's known performance issues [3].

The visited web pages are all known to the attacker in advance (assumption 8): In the literature of WF this assumption is referred to as the closed-world setting.

That is, a client is restricted to browse a limited number of web pages, monitored by the attacker and the attacker's goal is to identify the visited web page. This assumptions has been already criticized for being unrealistic [48, 33] since a client is unlikely to only browse a limited set of web pages and that set would be very small compared to the actual number of existing webpages. It is also unrealistic because all web traffic in Tor is sent through a single channel. Hence, the attacker is not given the destination of the traffic, he must assume that the channel could contain traffic to arbitrary webpages [41]. As a result, a more realistic open-world setting has been proposed [46, 45, 65] that allows the client to visit any page and the attacker's goal is to determine whether the client visits one of a small set of monitored pages.

Although the open world scenario is more realistic, the closed-world setting is still useful as it has been used in the literature to compare different attacks and defenses [66] and to give a lower bound on the defense effectiveness [34]. We also use the closed-world scenario in our work (see Chapter 4) to validate our hypothesis as we are not studying the performance of the WF in a realistic world scenario; rather, we evaluating a Tor browser configuration setting in light of web website fingerprinting attacks.

The visited web pages are only home pages of different websites (assumption 9): This assumption has been identified and studied by Miller *et al.* [41], where they differentiate between a website and a web page, as well as between index pages and non-index web pages (the internal web pages). They also emphasize the unrealistic nature of this assumption given that interactions with a website's home page alone ostensibly constitute a minority of user interactions with any given website. Moreover, it is advantageous for the attacker as the home pages for different websites are more distinguishable and generate more traffic than the internal web pages (i.e non-home pages) within a single website. They compare the accuracy of their own technique when identi-

fying traffic from website homepages to accuracy when comparing pages within a single website and they found that the accuracy is much lower comparing different web pages than comparing websites.

The visited web pages update their content rarely (assumption 10): There is extensive research regarding the type and frequency of changes to web sites, for example [16]. A key finding of those research efforts is the distinction between changes in terms of site structure, which occurs rather seldom, and content, which occurs frequently on many popular websites. This is reflected in the traffic traces and consequently in the accuracy of the WF attack. According to Herrmann *et al.* in [29], that is because website fingerprinting attacks exploit characteristics in the frequency distribution of the size of IP packets observed during transmission. This distribution is determined by the contents and structure of the HTML page and any embedded elements. Consequently, changes of the content may affect the fingerprint of the site, thus diminishing the accuracy of the classifier. We found several works in the literature to confirm that. Liberatore *et al.* [36] found larger delays between the training and test sets of traffic instances result in lower accuracy. Wang *et al.* [65] also observed that the web pages that instantly change their content are difficult to classify and their data collection process can't keep up with their content update. Morover, Juarez *et al.* [33] showed that the accuracy of classification decreases by 40 % in less than 10 days and further declines almost to zero after 90 days for Alexa Top 100 pages due to content change.

The victim is going to always request the localized version of the web pages (assumption 11): Although Panchenko *et al.* [46] were the first to point to the effect of the URL redirection generally on distributing the accuracy achieved on the WF classifiers, Wang [65] identified that such redirection in form of localization happens

in Tor because of the random selection of the exit node in Tor. This leads them to specify a localized version whenever possible for the accessed web pages to minimize the randomness of the exist node on their collected traffic. For example, they used localized (German) versions of the web pages in order to avoid different language versions [33]. The previous authors mentioned also this assumption is advantageous for the attacker as they noticed the total trace size of the English version for some web pages such as ask.com was about five times bigger than the German version of the same web page. As the WF attacker doesn't know what exit node a specific victim is using in a specific Tor circuit, he wouldn't be able to predict which version or language of a specific web page will be loaded by a specific user in case of the web pages that support localization.

Reasonable Assumptions We consider assumptions (1,2,3 and 12) to be reasonable. Here are our justifications for that.

The attacker can use similar Internet and Tor browser to the ones used by the victim (assumptions 1,2 and 3) We consider those three assumptions to be reasonable, because, in the real world, when the attacker targets a specific victim he can get a lot of background knowledge about the victim. For example, through physical contact or a long-term observation, the adversary can learn that she uses Tor Browser and may even infer the interesting list of websites that the user may visit with a high probability. He then can use Tor browser as configured [50] and recommended [60] by the Tor project.

The victim configured her browser to disable JavaScript while browsing (assumption 12): It is reasonable to assume the client (i.e. victim) to disable her browser's JavaScript plug-in to protect herself from the privacy and security concerns

related to the active content (embedded objects) as we explained in Section 2.2. This is true in case of Tor Browser too. However, surprisingly we found JavaScript is enabled by default in Tor Browser version 3.6.5 (this was the latest available version of Tor in the time of our work) which complies with Tor Browser design document [50]. When we consulted Tor Project website [60] to learn more about this configuration for JavaScript, we found that it is enabled by default for usability purposes only as we explained in Tor Browser design decisions related to the active content in Section 2.4.2.

The most relevant studies to the active content assumption imply that disabling JavaScript is advantageous for the WF attacker especially in case of the dynamic web pages fingerprinting. This is because dynamic web pages vary in the size and number of objects they contain based on the active content (JavaScript) configuration in a browser as described in Section 2.1.1. For example, when JavaScript is disabled the embedded objects in the web page will not be requested. Hence, the number of requests sent by the browser and the total number of packets returned by the server will be smaller. This makes the traffic pattern generated for that web page more similar and thus easier to identify than when JavaScript is enabled. Miller *et al.* [41] mentioned that active content is likely to complicate the traffic analysis attacks due to the increased generated traffic from web pages when the active content configuration is enabled on a PET under WF attack evaluation. They showed also how this browser configuration was enabled, disabled, or ignored in the studies they surveyed. They conclude with a recommendation to make this configuration in agreement with the threat model of the PET under analysis.

Another study conducted by Wang *et al.* [65] reported that the dynamic content, such as the web page content that changes based on the user location like Ads and other third-party content, lowers the accuracy rates of their SVM classifier. They noticed that the web pages that have dynamic content such as msn.com were among the most difficult

web pages to identify. They made, then, a relevant conclusion that the web designers can help with protecting from WF attacks by adding more randomized content to their web pages. Furthermore, in an earlier study [15], Cheng and Avnur in 1998 declared that dynamic web pages generated content of arbitrary sizes, which can make web page fingerprinting using their WF classifier an impossible task.

To sum up, disabling or enabling JavaScript are both reasonable assumptions given the fact the the enabled configuration is the default of the Tor Browser and the disabled one is the configuration recommend by Tor Browser. Disabling JavaScript is a advantageous for the attacker as it makes the classification task easier as it reduces the a mount of web page randomness.

3.4 Summary

In this chapter, we presented the needed background for the WF problem. We first introduced the WF attacks threat model where the attacker is a passive eavesdropper who locates himself between a targeted client using a web PET and the first proxy of the PET to identify the visited web page by the client over that PET. In case of Tor Browser, the attacker locate himself between the Tor Browser's client and the Guard node in Tor network. We then describe the steps included in a typical website fingerprinting attack which involves data collection, pre-processing training and testing.

Finally, we survey the existing work in the field of WF, categorizing it into three domains: website fingerprinting attacks; countermeasures that have been proposed against such attacks; and research in the practicality of these attacks. The WF attacks category shows how the web page loading process patterns has been used in the literature to identify the user's final destination (i.e the requested web server) despite her use of the web PETs. This category of survey also highlights Panchenko's SVM that we are going to use

later on our work. The WF defenses category then shows the different proposed defenses to counter those attacks by reducing the amount of the traffic information that is used in the fingerprinting process. It also shows that they generally have high cost in terms of added delay and bandwidth overhead, which make them impractical for deployment in real-world systems. That in turn, motivates the research in the practicality of the proposed attacks at the first place. We then survey that research by compiling the assumptions made in literature, categorizing them into reasonable and unreasonable based on Tor Browser threat model and Tor Browser design requirements. We also discuss the relevant previous studies for each one of the assumptions to support and justify our categorization.

In the next Chapter, we first formulate our hypothesis in light of our findings from the survey about the active content assumption in Tor Browser. We then discuss the set-up and implementation of our experiment to validate that hypothesis. We start by giving a description of the sources and formats of the datasets employed. We continue with the tools and platforms used in the development and implementation of our methods, and the necessary configurations required.

Part III

ADDRESSING THE PROBLEM

Chapter 4

Experimental Design

In Chapter 3, we concluded that the active content configuration recommended by Tor Project may make Tor Browser more vulnerable to website fingerprinting attacks than the default configuration. In other words, the active content recommended setting (i.e., disabling JavaScript) in Tor Browser is advantageous for the WF attacker, especially in case of the dynamic web pages fingerprinting. That is because dynamic web pages vary in the size and number of objects they contain based on the active content (JavaScript) configuration in a browser as we explained in Section 2.1.1. For example, when JavaScript is disabled, the embedded objects in the web page will not be requested. Hence, the number of requests sent by the browser and the total number of packets returned by the server will be smaller. Consequently, the amount of the randomized content included in the generated web traffic will be smaller too. This makes several traffic patterns generated from that web page more similar to each other and thus easier to identify than when JavaScript is enabled.

One way to test that hypothesis, is to apply a WF attack by following the procedure we described in Section 3.2 on a number of dynamic web pages visited by using Tor Browser configured in both the default setting (i.e., JavaScript is on) and the recom-

mended setting (i.e., JavaScript is off). We then compare the identification rates for the same web pages in both settings. If the identification rates are higher in the recommended setting, then our hypothesis is valid (i.e., disabling the active content makes Tor Browser more vulnerable to WF attacks).

To implement the steps involved in the WF attack procedure, we need to have a list of dynamic web pages and traffic dumps collected in the two mentioned settings of Tor Browser as defined in Section 2.4.2. After the data collection completes, the traffic dumps need to be used for training and testing a WF classifier. More specifically, we need to pre-possess those traffic dumps by converting them into traces, extract fingerprints from the traces for each one of the web pages and finally convert the extracted fingerprint into classification instances. After the classification instances are ready, we use a WF classifier to compare the identifiability of the web pages in both settings.

In the rest of this Chapter, we describe our experimental setup to implement each one of the steps described above considering the research assumptions in Section 3.3.3.1. Although most of those assumptions are not practical, the comparable website fingerprinting we surveyed in Subsection 3.3.1 and which we are building our research on have used them to model the real world scenario. In Section 4.1, we describe the limitations that prevented us from using the prior relevant works' datasets in our analysis. In the subsequent two sections, we describe the dataset we collected to avoid the aforementioned limitations. In Section 4.2, we describe our list of web pages by explaining how we select, clean and categorize its web pages. We then in Section 4.3, explain how we use that list in our data collection process. In Subsection 4.3.1, we describe the different software packages and libraries used in our data collection. We also describe how we configure each one of them. Furthermore, in Subsection 4.3.2, we describe how we implement our **Data Generator** script to orchestrate the data collection process among

the used software tools.

We further continue explaining our implementation for the remaining WF procedure steps. Section 4.4 describes how we clean the collected data from the corrupted traffic dumps and how we parse them into traces. After that, Section 4.6 describes briefly the extracted features from the traces along with Panchenko's SVM, which we used to classify them.

4.1 Prior Work Datasets

There are several datasets (lists of URLs and traffic dumps) that have been used in prior WF attacks in Tor [29, 46, 13, 65, 64, 33, 45]. However, none of the existing datasets allows an evaluation of the active content configuration effects on WF in Tor browser, though this constitutes an attack scenario to be expected in reality; a Tor user may use Tor Browser to visit static and dynamic web sites and a curious party may be interested in identifying or monitoring access to both types of websites. Table 4.1 gives a summary of the datasets used in the previous works. We name each dataset with the last name of the first author of the work concatenated with the last two digits of the year of the work's publication date. Work column describes the research paper where the relevant dataset was used, while Browser column indicates whether Tor Browser was used in the data collection or a Torified Firefox, which is a manually configured Firefox browser to use Tor. Both are described in more detail in Section 2.4. URLs column specifies whether the list of the used URLs were given or not. Dumps column refers to the traffic dumps generated from loading those webpages. Active Content column indicates whether JavaScript was enabled, disabled or just left to the default setting.

From that table we identify four specific limitations in those datasets. First, some of the prior studies [13, 65, 33] used Alexa [2], which is a well known list of most visited

Table 4.1: Summary of the datasets used in the WF attacks against Tor. The "X" indicates not available and the "✓" indicates available while the "?" indicates the authors didn't specify

Work	**PET**	**URLs**	**Dumps**	**Active Content**
Herrmann09 [29]	Torified Firefox	X	X	?
Panchenko11 [46]	Torified Firefox	X	X	Disabled
Cai12 [13]	Torified Firefox	Alexa	X	?
Wang13 [65]	Tor Browser	Alexa	✓	Default
Wang14 [64]	Tor Browser	list of blocked websites	✓	?
Juarez14 [33]	Tor Browser	Alexa	✓	Default
Panchenko16 [45]	TorBrowser	Multiple Resources	✓	?

URLs, but not necessarily a representative sample of the visited web pages by Tor users. Second, some of the datasets didn't include the URLs list used for evaluation, nor their type (i.e., whether they are static or dynamic), such as in Herrmann09 dataset [29] and Panchenko11 dataset [46]. Third, the other datasets Cai12 [13], Wang13 [65], Wang14 [64], and Panchenko16 [45] provided the URLs lists but didn't specify whether the browser active content property was on, off, or left to the default when they collected their traffic dumps. Finally, all the mentioned datasets above provided their collected web page traffic dumps in the trace representation that served their own purposes, instead of the raw format of their traffic dumps. Thus we could not use their provided traces in our evaluation.

To address the first two limitations (no representative web pages list or unspecified web pages type), we provide a new web page list and propose a web page categorization in Section 4.2. To avoid the other limitations related to the dumps representations and Tor Browser settings, we retrieve our own traffic dumps in Section 4.3

4.2 Our Web Pages List

As we are evaluating Tor browser, a more representative sample of web pages than than the web pages used in the previous studies would be a real list of web pages accessed through Tor network. For example, a Tor exit node operator can serve as a source of URLs. Recall that as we mentioned in Section 2.3.1, anyone can volunteer to be a Tor relay. Moreover, it is technically possible to modify the Tor source code in a similar way to what we did in our Tor client in Subsection 2.3.3 where we dump the Transport Layer Security (TLS) and Tor circuit keys to decrypt Tor traffic. Note that our patch only dumps keys if Tor source code is configured as a client and not as a relay, although it can be easily tuned to be used for that purpose too. Another way to accomplish that, is to install additional software to monitor or log plain texts that exit a specific Tor relay, and then extract the URLs for the visited web pages. A recent published work by Panchenko *et al.* [45] employed this method where they deploy a Tor exit node and then use it to compile their list of web pages.

However, obtaining such real web pages visited by real Tor users by using the methods described above could involve ethical [55] and legal considerations [58]. For instance, according to the Tor website [58] relay operators in the United States create civil and criminal liability for themselves under state or federal wiretap laws if they monitor Tor users' communications.

To avoid the potential ethical and legal issues, we built our better representative web pages list by selecting web pages from real world censored web pages; we then clean the list and finally, categorize the web pages of the list.

4.2.1 Web Pages Selection

One of the publicly available sources of real world censored web pages is the OpenNet Initiative (ONI). ONI is a collaborative partnership of University of Toronto, Harvard University and SecDev Group [27] to study and publicly report on the Internet filtering practices and control mechanisms around different countries of the world. For instance, it has lists of censored and filtered websites for about 61 countries around the world [44] (please refer to [44] to learn how they conducted their technical testing to build the lists for each country). We therefore believe ONI serves as a potent source of the URLs that can visited by a Tor user. To find a way to pick some of these countries, we use another publicly available report published by Reporters Without Borders (RWB) [51], which is an international, non-profit organization that promotes and defends freedom of information. The report is called "Enemies of the Internet" which draws attention to countries that disrupt the freedom of information with surveillance, and censorship (please refer to [51] to learn how they made their lists of countries). We use that report to pick randomly 17 of the countries that have been labeled as enemies of the Internet. After that, we look up those countries' profiles on the ONI's website. We use web pages from the 17 countries and classify them to one of four groups according to the categories suggested by ONI [44]. First, *Political*: This category is focused primarily on web sites that express views in opposition to those of the current government. They cover material related to the human rights, freedom of expression, minority rights, and religious movements. Second, *Social*: This group content is more related to sexuality, gambling, and illegal drugs, as well as other topics that may be perceived as offensive. Third, *Internet tools*: Web sites that provide Internet hosting, search and circumvention methods (such as Tor) are grouped in this category. The last group, *News sites*, includes popular news sites, such as BBC and CNN.

By using ONI and the RWB report, we manually construct an initial list of 130 unique URLs. We discuss next why and how we clean this list.

4.2.2 Web Pages Cleaning

They are three cases where the web pages URLs can lead to corruption when we use them for our data collection: web page unavailability, web page redirection [46] and web page localization [46]. Web page unavailability (i.e., a web page is no longer exists) and web page redirection (i.e., a web page redirects to another URL) cause an automated crawler, such as Selenium that we use in Section 4.3) to consider a targeted URL as loaded even though it has not. Rather, what is loaded is information about the web page unavailability or re-direction to another URL. Localization also causes one web page's URL to direct different instances of the same web page based on the location of the web client (which is in case of Tor is the exit node as we explained in Subsection 2.3.1). For example, when the exit node is in UK, a Tor Browser request to **bbc.com/news/**, would return the localized version **bbc.com/news/uk**. Any one of those three cases, leads to a misrepresentation of a targeted URL. That is, the retrieved web page is not the web page corresponding to requested URL. It is instead an unavailability message, a redirection message or a totally different instance of the targeted URL. This consequently leads to a degradation of the accuracy rates of the classifiers that use those corrupted traffic instances.

To avoid degradation of accuracy rates due to that corruption in our dataset, we clean the list of our URLs to make sure that each accessed web page represents the intended URL to be visited. To accomplish that, we use Tor browser to manually load each web page from the list one by one. We then remove web pages that were unavailable at the time when we accessed them. We also disregard those with a redirect message

and add the URLs of the retrieved web pages instead as Panchenko *et al.* did in [46]. To handle the localization cases, we manually specified a localized version of the web pages as Wang *et al.* did in [65]. That is, instead of using bbc.com/news/ in our list, we use bbc.com/news/uk. We believe that is a reasonable solution, as an attacker who is interested in identifying the different versions of a web page, would be able to treat each version of the web page as a separate web page and train their classifier accordingly.

After completing the cleaning process, our initial list of web pages cut down to 100 unique URL out of the 130 URL. We call this list **CensoredURLs** list and we provide it in Appendix A. Most of the web pages we tested were not available at the time of testing and this may be due to the fact they are censored web pages as they could be taken down by some organizations or authorities.

4.2.3 Web Pages Categorization

To get a set of dynamic web pages out of that list. We use the definition for dynamic web pages based on the web technologies that have been used in building them as introduced in Section 2.1.2. One way to implement that definition is by leveraging the fact that Tor Browser uses Firefox, which has a powerful add-ons system and also allows users to install plug-ins that augment the browser's functionality. In particular, we use **Adblock Plus** [70], which is a well known add-on to detect and block Ads in a given web page, to detect the number of embedded Ads. We also use **Wappalyzer** [24], which is a browser add-on that detects the technologies used to build the given web page, to learn about the JavaScript frameworks used in a specific web page.

We install Adblock Plus and Wappalyzer add-ons in Tor Browser. We then manually retrieve the web page that we want to categorize. While retrieving it we take note of the number of the ads reported by the Adblock Plus (let's call it **AdsNum**) and the number

of the JavaScript frameworks reported by Wappalyzer (let's call it **ScriptNum**). We then find out if the sum of AdsNum and ScriptNum equals 0. If it is 0, then we consider the web page not dynamic (it is rather a static one). If it is not 0, then we check if it is smaller than three; if so we categorize the web page as a **light** dynamic web page. If it is bigger than 3, we consider that web page as a **heavy** dynamic web page.

By using that methodology, we can get a rough idea about the dynamism of a specific web page by categorizing it to static, light dynamic and heavy dynamic. However, it is currently a manual method, so it would take a lot of time to go over all the web pages in our CensoredURLs list. As a result, we decided to take a smaller set of web pages (10 web pages) from the 100 list to investigate. We ended up working with the web pages listed in Table 4.2; we call this sublist as **CensoredDynamicURLs**.

We think our implementations for web page categorization is not the best way to learn about a web page dynamism but we think it is sufficient for our research purposes and our time concerns as we augmented Tor Browser with tools that give us indicators about the dynamism of web pages. We discuss this more in our research limitations in Section 5.4.

4.3 Data Collection

In order to generate traffic dumps from the CensoredDynamicURLs list, we use **Traffic Generator**, which is a Python script we implement to automate web pages loading and traffic recording. It takes as input a list of web pages to generate traffic from, and the number of visits for these web pages. It then generates as output the dump files which correspond to each visit and set of related logs. The Traffic Generator works by controlling five software components. Figure 4.1 shows those different components along with the Traffic Generator script that controls them. The components and their

Table 4.2: CensoredDynaimcURLs list

No.	URL	Type
1	https://www.anonymizer.com/	Tools
2	https://kproxy.com/	Tools
3	https://english.alarabiya.net/	News
4	http://www.bbc.com/arabic	News
5	https://zh-cn.facebook.com//	Socail
6	https://twitter.com/?lang=zh-cn	Socail
7	http://www.x-cafevn.org/	Political
8	http://www.mesfinwoldemariam.org/	Political
9	http://www.alduraz.net/	Political
10	https://www.amnesty.org/en/	News Sites

configurations are explained in Subsection 4.3.1 and how the Traffic Generator controls them to collect data is discussed in Subsection 4.3.2.

4.3.1 Software and Libraries

The Tor Browser, which is a self-contained software package, combines a pre-configured Tor client and a stand-alone web browser based on Mozilla Firefox as we introduced it in Section 2.4.1. We strive to apply as little change as possible to the Tor Browser configurations so our simulated traffic generator stays as close as possible to the configurations of a real world Tor Browser user. To that effect, we only change the default start up page and the browser automatic update configuration. We set Tor Browser's home page to blank instead of its default web page to avoid its generated traffic from interfering with the traffic of the targeted web page retrieved over Tor Browser. We also disable the automatic update to prevent the generated traffic from loading the update form interfering

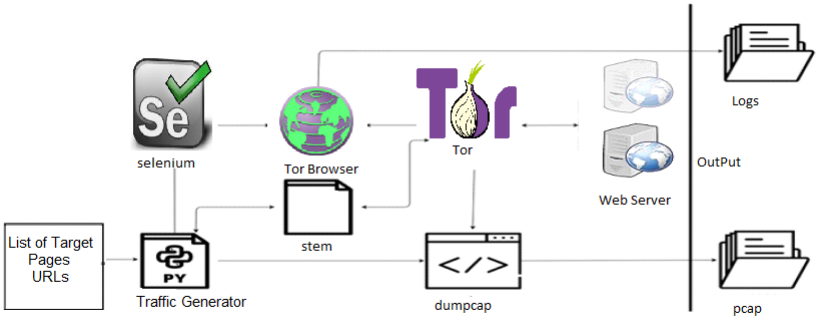

Figure 4.1: Traffic Generator Components

with the target web page traffic.

We then configure Tor Browser to implement the two settings for our hypothesis: one with the enabled active content (i.e., default setting) and the other one with the disabled active content (recommended setting). We control all of configurations we need by changing Tor Browser relevant preferences, which can be displayed by typing about:config in the address bar of Tor Browser [42]. We use Tor Browser version 3.6.5 which includes a modified version of Mozilla Firefox 24.8.0esr and a bundled Tor version 0.2.4.23.

Tor is the Tor software package that acts as an OP in the user machine (as explained in Section 2.3.1). In our data collection, we didn't use the bundled Tor that comes with Tor Browser. Instead, we use our patched Tor version of Tor described in Subsection 2.3.3, which we further configure to use unlimited set of guard nodes so the collected data will be more representative of the entire Tor network as Wang *et al.* did in [65]. Recall from Section 2.3.1 that Tor picks a guard relay from a list of three guard relays, lasting for 30 to 60 days for the same client. By disabling the entry guard list configuration in Tor, we allow every entry relay in Tor network to be a guard node every time Tor builds a new circuit. We use the configuration manual provided by Tor in [20]. We set this

configuration value by using Stem which is described next.

Stem is a Python implementation of the Tor Control Protocol to control functionality of Tor [56]. We use it to pass the entry guard configuration value that we just described. We also use it to start, restart Tor and close streams after we finish loading a webpage. However, our Tor controller does not interfere with the default circuit creation as in previous work [65], assignment and fetching of websites. We use version 1.4.

Selenium-WebDriver API is a browser automation software [53]. We use it to mimic a user's visits to a specific web page. We use Firefox webDriver to control Tor Browser preferences as explained in the Tor Browser component. We also configure the webDriver by changing the hard visit timeout from never to 60 seconds as Juarez *et al.* did in [33]. That means if a specific web page doesn't start loading within 60 seconds, through and exception in Selenium, we then initiate a reload command just as what a real user would do when a web page fails to load for some reason. Otherwise, Selenium will wait forever, waiting for that page to load.

Dumpcap is a network traffic sniffing tool that captures packet data from a live network and writes them to a packet capture file called pcap file [69]. We use it for traffic recording while visiting web pages by Selenium. We configure Dumcap with a set of configurations to capture traffic on a wired interface for a specific duration of time and to capture the full packet length. We use the manual configuration in [69]. We also set up filters given in the manual [72] while capturing to capture only the traffic coming from and leaving the system where we have our Tor browser running. To facilitate isolating Tor Browser related traffic further, we also configure the whole hosting operating system to reduce the known background activities that generate web traffic. As these details

are specific to the operating system settings that we use and are not directly related to Tor or Tor browser, we leave this technical details without further description. It is also worth mentioning that we have both Dumpcap and Tor Browser in the same machine so we follow the set up instructions provided by Wireshark guide in [40]. We use Dumpcap version 1.12.1.

4.3.2 Data Generator Script

The Data Generator script orchestrates data collection flow between those components. It starts by taking as inputs the number of traffic examples that need to be collected along with the list of the URLs to be retrieved. It collects the traffic dumps in batches; each batch presents a single visit to each one of the web pages in the list. The reason for that is to prevent our data collection from being unrealistically advantageous for the classification process by training and testing traffic instances that belong to to the same Tor circuit. That crosses out the inherent difference of circuits which imposes a difference upon the traffic instances of different web pages, which could help the classifier separate them as pointed by Wang *et al.* [65]. This happens in Tor by default as we explained in Subsection 2.3.1. Recall that Tor only uses a circuit for up to 10 minutes, after which Tor will not launch new connections on that circuit. If that configuration is maintained, and if web pages are visited consecutively without an explicit way to change the circuit, then several traffic dumps maybe accessed with the same circuit. As a result, the traffic instances, which are used in training and testing the classifier, may belong to the same circuit. That is an unrealistic advantage to the attacker because it is highly unlikely the attacker can collect training instances from the same Tor circuits used by the Tor victim. To ensure that we never use the same circuit for both training and testing for a specific web page, we collected our traffic dumps in batches. A batch contains a traffic dump

that is generated from a single visit for each one of the web pages in the given list of URLs.

While collecting a specific batch, the script connects with Stem to start Tor. It then applies the settings for Selenium and the specified profile for Tor Browser to use. After starting the Tor Browser with a specific profile, the Data Generator starts iterating over the list that contains the URLs to be retrieved and captured. A capture of a web page is taken by first starting Dumpcap and then issuing a Selenium command to retrieve a URL from the list. After Selenium reports if the page has been loaded, an additional 20 seconds sleep takes place to roughly correspond to a user who spends a small amount of time after the web page loads. After this pause time, the script issues a stop loading command to Selenium. It issues an end capturing command to Dumpcap as well. Next, the scripts commands Stem to close all the open circuit streams. Another pause for 4 seconds takes place just to make sure that the closed streams are all closed, so the traffic generated from them would not affect the next traffic recording process as Wang *et al.* did in [64]. The script then closes the opened Tor Browser just to make sure the memory caching which is enabled by default in Tor Browser (as we discussed in Sub-subsection 3.3.3.1) won't effect the loading process of the next web pages. After the script finishes capturing all URLs in the list, it then starts a new batch again until it collects the required number of the traffic examples.

We implement Data Generator script in away that handles the time out errors that may happen while loading a web page. It also handles the other possible components failures such as Tor connectivity problem or the sniffer recording problem. It does that in a way that makes it produce at the end of each iteration a complete batch that has exactly one traffic file generated from each URL in the list.

4.4 Traffic Pre-Processing

We process the collected traffic dumps (i.e., pcap files) with the Data Generator in three steps: Checking if for every web page and every batch, there is a pcap file; identifying and handling the invalid pcap files; parsing the valid ones to obtain the corresponding traces, so they can be ready for use in the next features extraction phase.

For the first two steps, we wrote a Python script to iterate over all the batches and read each pcap individually. It then reports if these files could not be opened for further processing (such as, corrupted pcap files). It then deletes the whole batch where the corrupted file has been found. This script cleans our dataset from incomplete batches and invalid pcap files. We share its code publicly in [4].

The final step in our data processing is to parse the pcap files from the raw format to traces. That is, extracting each packet length, direction (from or to the server) and time. For the parsing, we use Panchecko's parsing method [46] and Dyer's implementation [23] for it. We pick this parser as we are using these two works already in our classification stage. However, we needed to adjust the parser to suit our data collection method and we implement our own traffic filters to only extract the packets that belong to the conversation between the Tor client and between the used guard node in the circuit. Our modified parser is available in [4].

4.5 Features Extraction

To extract fingerprints (identifiable features) from the traces we obtained from the previous Section, we use the features provided by Panchenko *et al.* [46]. They used a wide variety of coarse and fine traffic features of the network data mostly derived from packet lengths and ordering. Some of these features include the total number of bytes transmit-

ted, total number of packets transmitted, proportion of packets in each direction (i.e., from the client to the server and vice versa), and raw counts of packet lengths (i.e., packet sizes). There are also several features known as markers that delineate when information flow over the encrypted connection has changed direction. These markers aggregate bandwidth and number of packets into discrete chunks.

To implement the feature extraction in our data, we use Dyer's Framework as it has an implementation for Panchenko's classifier which also includes an implementation for their features as the features are then used to prepare the instances that will eventually be used by that classifier.

4.6 Classification

In order to compare the identifiability of the dynamic web pages in the two settings we pick Panchenko's classifier, which is a SVM classifier with Radial Basis Function (RBF) kernel that has been configured and optimized by Panchenko *et al.* in [46]. It was the first successful WF attack against Tor, where the authors managed to achieve an accuracy rate up to 55%. We use their classifier along with their set of features and parameters to test our hypothesis. An SVM is a supervised learning method that classifies points in a high-dimensional space by determining their relation to a separating hyperplane. In particular, the SVM is trained by providing labeled points (traffic instances in case of WF) and discovering a hyperplane that maximally separates the two classes (i.e., web pages) of points. Classification occurs by determining where the point in question lies in relation to the splitting hyperplane.

We applied Panchenko's SVM implementation provided by Dyer's Framework [23]. It is a highly modular Framework that provides an implementation for a number of the website WF attacks and defences that we describe in our survey in Section 3.3. It is

also written in Python which is the same programming language we use to implement our data collection script in Subsection 4.3.2. It implements Panchenko's classifier as configured by Panchenko *et al.* [46] by using the same RBF kernel with parameters: cost of errors if no perfectly separating hyperplane can be found; the vectors which are on the wrong side of the plane are multiplied by $C = 2^{17}$; and kernel parameter that determines the smoothness of the hyperplane, (i.e., size of the region influenced by a support vector) $\lambda = 2^{-19}$. An interested reader is pointed to [46] for thorough information about their SVM.

4.7 Summary

In this Chapter, we first highlighted our hypothesis in light of the conclusions we made from our survey. That is, the active content recommended setting may make Tor Browser more vulnerable to WF attacks. We then discussed our research methodology to test that hypothesis by applying WF procedure on a set of dynamic web pages in both settings and then comparing their identifiability in both settings. Next, we discuss our implementation decisions by describing the software tools, frameworks and the necessary configurations required in each step of the WF procedure: data collection, prepossessing, feature extractions and classification.

In the next Chapter, we use our dataset, our data collection script, and the extended Dyer's traffic analysis framework to validate our hypothesis. We first outline the evaluation criteria employed, while the remaining sections present the results of our experiments, an analysis of the observed results and our research limitations.

Chapter 5

Evaluation and Experimental Results

In Chapter 3, we concluded that the active content configuration recommended by Tor Project may make Tor Browser more vulnerable to WF attacks than the default configuration especially in case of the dynamic web pages. Chapter 4 discussed our research methodology, our collected dataset, and the experimental setup and implementation to test that hypothesis.

In this chapter, we provide the results of our experimentation of identifying a set of dynamic URLs in the recommended Tor setting and the default setting. We begin with an overview of the evaluation metric used in our experimentation. We then present the obtained results for the classification.

5.1 Evaluation Metric

Before we can use our dataset CensoredDynamicURLs, which is described in Subsection 4.2.3, we need to split it into *training* and *test* data. We use as in Panchenko *et al.*

[46], ten-fold stratified cross-validation in order to obtain representative and comprehensive test and training sets from the collected data. Cross-validation [28] is a common method employed in data mining. It is often used for small datasets where the data is split evenly into n smaller parts (each part called a fold). The whole process of training and testing is then repeated n times, using one of the n folds as test data, and the remaining n - 1 folds as training data in turn. After that, the results are averaged, thereby providing a more evenly distributed inference from the given dataset. In our evaluation, n is set to 10 with 2 instances per fold and per class. Stratification ensures that the instances within a class are represented in as balanced a way as possible in each fold. If not stated otherwise, we used 18 instances for each censored page as training and 2 for testing, applying the support vector classification on the features as we explained in Section 4.5.

We then performed the recognition (identification) using Panchenko's SVM classifier described in Section 4.6 for our datasets in both settings, using the recommended (R) and the default dynamic setting (D). No optimization was done for the SVM parameters or for the used feature sets. The SVM then determines a score for each test instance with respect to all classes, and assigns the test instance to the class with the highest score. We determine the detection rate (accuracy) as the percentage of correctly classified instances divided by all instances. We show the obtained accuracy results next.

5.2 Evaluation Results

Our evaluation results show that the web pages in the recommended setting (i.e., JavaScript is disabled) generally has higher accuracy rates than the default setting (i.e., JavaScript is enabled). Higher accuracy means the web pages are more identifiable in the recommended setting than the default setting. Table 5.1 shows the obtained results.

Table 5.1: The Accuracy results for CensoredDynaimcURLs (%)

Webpages Set	**Recommended Settings(R)**	**Default Settings(D)**
Light Dynamic	71.11	68.89
All	66.67	61.3
Heavy Dynamic	42.22	35.55

Recall that *light dynamic* refers to web pages that have fewer than 3 Ads and JavaScripts components, as we described in Section 4.2.3. While heavy dynamic web pages have more than 3. We use that categorization mainly as rough indicator to the dynamism of a web page. Using that categorization we classify our CensoredDynamicURLs shown in Table 4.2 as the following: pages with index number (1,3,4,8,10) are light. Web pages with index number (2,5,6,7,9) are heavy. *All* refers to all 10 web pages in the same table.

A first look to the table shows that while dynamism amount increased from the top to the bottom and from the left to the right, the accuracy decreased. So we have the least amount of the dynamic content in case of the light web pages in the recommended setting where we have the highest accuracy 71.11%. We then have the most amount of the active content in the heavy dynamic in the default setting where we get the lowest accuracy 35.55%.

We also can see that accuracy in the recommended setting is higher than the default settings, where the accuracies are 71.11%, 42.22%, 66.67% compared to 68.89%, 35.55%, 61.3% for the light Dynamic, heavy Dynamic, and all respectively. In other words, the accuracy for the heavy dynamic web pages was higher in the Recommended setting than in the default setting. This was true also in the case of the light dynamic web pages dataset that we tested. However, the accuracy for the light dynamic pages was much higher than the accuracy for the heavy dynamic pages overall.

Having higher accuracy in the light dynamic web pages, confirms that increasing the dynamism of the web pages makes them harder to identify. This makes intuitive sense because the dynamic content usually changes randomly; for example, websites that have news that update continuously, and websites that have embedded ads. The current existing classifiers used in the website fingerprinting attacks does not take into consideration the randomness that may take place on the traffic generated from the web pages.

5.3 Comparison with the existing works

We couldn't directly validate if the dynamic web pages, that have been evaluated in the relevant existing work (that we described in Chapter 3), are easier to identify in the recommended setting than the default settings. This is because we were unable to find the necessary data needed to make that comparison. We, however, showed in Table 4.1 that the already existing work [65] reports that some of the web pages (such as bbc.co.uk and msn.com) result in notably lower accuracy than the rest of the other web pages in their study.

5.4 Limitations

In full acknowledgement of the importance of presenting the limitations of our work, we consider here three types of limitations: those affecting our collected data, the performance of the classifier employed, as well as the issues not considered in our work.

Collected data: There are certain limitations regarding our collected data which relates to the categorization of our URLs as *static* and *dynamic*. We categorized our URLs

as dynamic or static manually by using two browser plugins: Wappalyzer [24] to show the web technologies used to build a specific website, and Adblock Plus plug-in [70] to learn about the number of Ads embedded in a specific web page. Using that manual way to learn about the web pages made it easier and faster for us to put a smaller number of web pages under monitoring and focus on our main research hypothesis within our scope and time constraints. However, we recognize that using a manual method limited teh number of web pages we were able to test, and so a more automated means of categorizing the data would be preferable.

Classification: There were no optimizations applied at the classifier level or the feature level used for obtaining our accuracy results. While we believe this was sufficient for the purposes of our current research, we acknowledge there is possible room to improve the accuracy we achieved, by finding more powerful features related to the dynamic web pages, or stronger classifiers that can perform well on dynamic content.

General applicability: We also recognize that the above mentioned limitations (with respect to the collected data and classification methods used) resulted in a consequent narrowing of the work scope. This directly impacts the immediate applicable generalization of our work.

5.5 Summary

In this chapter, we first described the evaluation metric used. We then presented the results of our experiments arising from our preliminary evaluation of the recommended Tor setting for the active content in light of WF attacks. From the results, the recommended setting for the active content in Tor Browser may not be as safe as it seems, as

it makes web pages more identifiable than the default setting. We then concluded by discussing the limitations of our work.

While we demonstrate the persisting effects of WF attacks on *dynamic* web pages in our preliminary study, we fully acknowledge that our work should be extended and generalized to larger web page datasets, with more optimized classification algorithms which will be able to better account for more of the varying aspects that contribute to the true dynamism of modern web pages.

In the following final Chapter 6, we present the concluding remarks, discuss our contributions, and then give a brief overview of our future work.

Chapter 6

Conclusions

Tor Browser is a PET to provide private and anonymous web browsing as it enables users to hide content and addresses of requested web pages from external observers. However, this protection is endangered by WF attacks that allow an external, passive attacker between the Tor client and the guard node to uncover the identity of the requested web pages.

In this thesis, we have shown that a configuration recommendation provided by Tor for the privacy and anonymity of users of the Tor Browser may be less safe than the default setting, especially when considering existing website fingerprinting attacks. In particular, we showed that disabling JavaScript in Tor Browser, as recommended, may make the user more vulnerable to the website fingerprinting attacks in surveillance and censorship settings.

While only using the already existing proposed classifiers and publicly available information about a set of censored webpages, we achieve a recognition rate of 42% when JavaScript is disabled, compared to 35% when turned on. Our results imply that a moderately resourced attacker can successfully identify the final destination of a Tor browser user. This builds a case against the privacy and security assurance claimed by Tor (a

PET that promises to strengthen users' civil rights such as privacy and freedom of speech around the world). Our finding shows that this Tor recommendation may potentially harm users who rely on Tor to evade censorship. Specifically, it makes the attacker classification task easier as he doesn't need to take into consideration the randomness of web pages while building classifiers to identify the visited web pages.

Through analyzing the Tor specification, source code, Tor browser design documents, and recommendations, we set up an environment to record traffic for a set of real censored web pages. We then built a Python traffic analysis framework that allows us to sanitize and parse our data. We also successfully carried out major code re-factoring for a nine year-old Tor dissector that we are working on with the Wireshark development team to integrate into their Wireshark baseline code. We also patched Tor source code to dump the keys used by Tor client. While this allows the decryption of web traffic (which is out the scope of the website fingerprinting attacks), we believe it could be useful as an educational tool for the Tor protocol. We then used Dyer's Framework to classify and determine how accurately an attacker can distinguish between web pages instances.

In light of our finding, we believe it is important to reconsider the risks of the recommendation made by Tor regarding its JavaScript settings. Our finding shows that there is not only a trade-off between the security and usability related to this configuration, but also a security and privacy trade-off as well. That is, disabling JavaScript may make the user safer against the well known security and privacy leaks related to the active content, but, it makes the user more vulnerable to the existing website fingerprinting attacks. We therefore strongly support Tor's alternative design requirement to sandbox JavaScript (mentioned in [50]), as a possible alternative solution to disabling JavaScript. This is quite important in the face of several studies [22, 34] that show that there are is no effective deployable defense yet for this type of traffic analysis attacks, as was presented

in our survey in Chapter 3.

In conclusion, we have found a Tor recommendation for a configuration that makes Tor browser more vulnerable to website fingerprinting attacks. To our knowledge, we are the first to demonstrate this. We have also brought new insight into the extra threat posed by the JavaScript settings in Tor Browser, which will hopefully result in more attention to this configuration by Tor browser teams and users. We also tried to provide a more realistic analysis (although preliminary one) for the website fingerprinting attacks in Tor by considering a setting that is usually turned off by default in other website fingerprint attacks studies. Finally, we are publicly sharing the tools we developed to advance the ongoing research and discussion in this important field of research [4].

6.1 Future Work

Regarding possible future work, we see several possible directions based on our survey presented in Section 3.3 and our work limitations presented in Section 5.4.

One main direction of advancing the field of WF attacks in light of the survey, is in the consideration to evaluate WF attacks against Tor Browser in more realistic and practical settings. One way to accomplish that, is to build classifiers that take into consideration the list of the assumptions we compiled in Subsection 3.3.3.2. This eventually help Tor users, Tor stakeholders and the research community to make a more realistic assessment of the threat they are facing.

Another area of concentration lies in considering limitations related to our work. For example, expanding our experiment to make it applicable to a larger number and variety of web pages by finding a better, automated ways to categorize the web pages into static and dynamic. It also can be expanded by employing a stronger classifier that can work with the dynamic web pages just as there are several classifiers in the literature that give

high accuracy rates in the case of static web pages.

A final avenue of importance is in exploring the possible effect of JavaScript sandboxing on the website fingerprinting attacks.

Appendix A

CensoredURLs List

Political

1. http://www.pajk-online.com
2. http://we-change.org/
3. http://www.roozmaregiha2.blogfa.com/
4. www.libya-al-mostakbal.org
5. http://x-cafevn.org
6. www.asharqalawsat.com
7. http://www.alhadath-yemen.com/
8. www.pajk-online.com
9. www.circumcision.org
10. www.answering-islam.org
11. www.yahosein.com

12. http://www.tayyar.org/
13. www.akhbar-libyaonline.com
14. www.qatarsucks.com
15. http://www.hrichina.org
16. http://vietnamdaily.com
17. www.bahrainrights.org
18. http://www.kyrgyzpress.com
19. www.annahar.com .
20. http://danviet.com.au
21. www.jdl.org
22. http://www.alduraz.net/
23. http://www.mingpao.com
24. https://www.article19.org
25. http://freespeechcoalition.com
26. http://www.alquds.co.uk
27. http://www.mesfinwoldemariam.org
28. http://www.fdtl.org
29. http://www.moncefmarzouki.net

30. https://nawaat.org/portail
31. http://www.tunezine.com
32. https://www.amnesty.org/en/
33. https://en.rsf.org
34. http://www.ihrc.org.uk
35. http://anhri.net
36. http://taiwandc.org/
37. http://tiananmenvigil.org
38. http://friendsoftibet.org/

Social

1. http://www.bebo.com/
2. http://www.photobucket.com/
3. https://www.misstravel.com/
4. http://www.ikhwan.net/
5. http://facebookviet.com
6. http://www.arablounge.com/
7. http://www.jasadmag.com
8. http://www.xnxx.com/

9. http://vnyouth.com
10. circumcision.org
11. sexualhealth.com
12. www.lesbians-against-violence.com
13. www.bglad.com
14. https://www.tmdhosting.com/
15. http://facebook.com
16. https://www.seekingarrangement.com/
17. www.collegehumor.com
18. www.arabtimes.com
19. www.blogger.com
20. www.online-dating.org
21. www.metacafe.com
22. www.breastenlargementmagazine.com
23. www.gayonthenet.net
24. http://www.luxuryjoy.com/
25. www.lesbiansubmission.com
26. http://www.thequran.com/

27. http://www.islameyat.com/
28. www.mySpace.com
29. www.orkut.com
30. www.Flickr.com
31. http://www.xvideos.com
32. http://xhamster.com
33. http://www.nakednews.com
34. www.delicious.com
35. www.pornhub.com
36. http://www.888casino.com
37. https://www.partypoker.com
38. www.twitter.com
39. www.facebook.com
40. http://www.youporn.com

Internet Tools

1. https://www.anonymizer.com
2. https://www.torproject.org
3. http://www.guardster.com

4. https://www.avoidfilters.com
5. http://www.vobas.com
6. www.multiproxy.org
7. http://peacefire.org/
8. http://gardennetworks.com/
9. https://kproxy.com/
10. http://bypassrestrictions.com/

News Sites

1. http://www.alarabiya.net
2. http://www.huffingtonpost.com
3. http://www.bbc.co.uk
4. www.radiozamaneh.com
5. http://www.roozonline.com
6. http://boxun.com/
7. http://www.epochtimes.com
8. http://www.bbc.co.uk/news/world/
9. http://www.mingpao.com
10. http://www.rfa.org/english/

11. https://globalvoices.org

12. http://www.dailymotion.com

References

[1] Firebug. http://getfirebug.com/whatisfirebug]. (Accessed on 07/28/2017).

[2] Alexa Internet, Inc. The top 500 sites on the web . http://www.alexa.com/topsites. (Accessed on 1/08/2017).

[3] Alsabah, M., and Goldberg, I. Performance and Security Improvements for Tor: A Survey. *ACM Comput. Surv. 49*, 2 (Sept. 2016), 32:1–32:36.

[4] Alshammari, Fayzah and Adams, Carlisle. Tor Traffic Analysis Framework. https://github.com/am5april/AdjustedFramework. (Accessed on 1/08/2017).

[5] Bauer, K. S. *Improving Security and Performance in Low Latency Anonymous Networks*. PhD dissertation, University of Colorado at Boulder, 2011.

[6] Bennett, C. J. cookies, web bugs, webcams and cue cats: Patterns of surveillance on the world wide web.

[7] Berthold, O., Federrath, H., and Köpsell, S. Web MIXes: A System for Anonymous and Unobservable Internet Access. In *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability* (New York, NY, USA, 2001), Springer-Verlag New York, Inc., pp. 115–129.

[8] Bissias, G. D., Liberatore, M., Jensen, D., and Levine, B. N. Privacy Vulnerabilities in Encrypted HTTP Streams. In *Proceedings of the 5th International Conference on Privacy Enhancing Technologies* (Berlin, Heidelberg, 2006), PET'05, Springer-Verlag, pp. 1–11.

[9] C. V. Wright, L. Ballard, F. M., and Masson, G. M. Language Identification of Encrypted VOIP Traffic. In *Proceedings of USENIX Security, Boston, MA, USA, August 2007* (2008).

[10] C. V. Wright, L. Ballard, S. C. F. M., and Masson, G. Spot me if you can: Uncovering Spoken Phrases in Encrypted VOIP conversations. In *Proceedings of IEEE SP, Oakland, California, USA, May 2008* (2008).

[11] Cai, X., Nithyanand, R., and Johnson, R. New approaches to website fingerprinting defenses. *CoRR abs/1401.6022* (2014).

[12] Cai, X., Nithyanand, R., Wang, T., Johnson, R., and Goldberg, I. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2014), CCS '14, ACM, pp. 227–238.

[13] Cai, X., Zhang, X. C., Joshi, B., and Johnson, R. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (New York, NY, USA, 2012), CCS '12, ACM, pp. 605–616.

[14] Chen, F., and Pasquale, J. Toward Improving Path Selection in Tor. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010* (Dec 2010), pp. 1–6.

[15] Cheng, H., and Avnur, R. Traffic Analysis of SSL Encrypted Web Browsing. *URL citeseer. ist. psu. edu/656522. html* (1998).

[16] Cho, J., and Garcia-Molina, H. Estimating Frequency of Change. *ACM Trans. Internet Technol. 3*, 3 (Aug. 2003), 256–290.

[17] Comer, D. *Computer Networks and Internets*, 5th ed. Prentice Hall, 2009.

[18] Díaz, C., Seys, S., Claessens, J., and Preneel, B. Towards Measuring Anonymity. In *Proceedings of the 2nd International Conference on Privacy Enhancing Technologies* (Berlin, Heidelberg, 2003), PET'02, Springer-Verlag, pp. 54–68.

[19] Dingledine, R., Mathewson, N., and Syverson, P. Tor: The Second-generation Onion Router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13* (Berkeley, CA, USA, 2004), SSYM'04, USENIX Association, pp. 21–21.

[20] Dingledine, Roger and Mathewson, Nick. Tor Dev Manual. `https://www.torproject.org/docs/tor-manual.html.en`. (Accessed on 1/08/2017).

[21] Dutton, W., Dopatka, A., Hills, M., Law, G., and Nash, V. *Freedom of connection, freedom of expression: the changing legal and regulatory ecology shaping the Internet*. UNESCO, 2011.

[22] Dyer, K. P., Coull, S. E., Ristenpart, T., and Shrimpton, T. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2012), SP '12, IEEE Computer Society, pp. 332–346.

[23] Dyer, Kevin P. Traffic Analysis Framework. `https://github.com/kpdyer/website-fingerprinting`. (Accessed on 1/08/2017).

[24] Elbert, A. Wappalyzer :: Add-ons for Firefox. `https://addons.mozilla.org/en-US/firefox/addon/wappalyzer/`. (Accessed on 06/07/2016).

[25] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. Hypertext Transfer Protocol – HTTP/1.1. RFC Editor.

[26] Gilad, Y., and Herzberg, A. Spying in the Dark: TCP and Tor Traffic Analysis. In *Proceedings of the 12th International Conference on Privacy Enhancing Technologies* (Berlin, Heidelberg, 2012), PETS'12, Springer-Verlag, pp. 100–119.

[27] Gill, P., Crete-Nishihata, M., Dalek, J., Goldberg, S., Senft, A., and Wiseman, G. Characterizing Web Censorship Worldwide: Another Look at the OpenNet Initiative Data. *ACM Trans. Web 9*, 1 (Jan. 2015), 4:1–4:29.

[28] Han, J., Pei, J., and Kamber, M. *Data Mining: Concepts and Techniques*. Elsevier, 2011.

[29] Herrmann, D., Wendolsky, R., and Federrath, H. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial NaïVe-bayes Classifier. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security* (New York, NY, USA, 2009), CCSW '09, ACM, pp. 31–42.

[30] Hintz, A. Fingerprinting Websites Using Traffic Analysis. In *Proceedings of the 2nd International Conference on Privacy Enhancing Technologies* (Berlin, Heidelberg, 2003), PET'02, Springer-Verlag, pp. 171–178.

[31] Jana, S., and Shmatikov, V. Memento: Learning Secrets from Process Footprints. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2012), SP '12, IEEE Computer Society, pp. 143–157.

[32] JAP. JAP – ANONYMITY & PRIVACY. `https://anon.inf.tu-dresden.de/index_en.html`. (Accessed on 06/07/2014).

[33] Juarez, M., Afroz, S., Acar, G., Diaz, C., and Greenstadt, R. A Critical Evaluation of Website Fingerprinting Attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2014), CCS '14, ACM, pp. 263–274.

[34] Juárez, M., Imani, M., Perry, M., Díaz, C., and Wright, M. WTF-PAD: toward an efficient website fingerprinting defense for tor. *CoRR abs/1512.00524* (2015).

[35] Kwon, A. *Circuit Fingerprinting Attacks: Passive Deanonymization of Tor Hidden ervices*. PhD thesis, Massachusetts Institute of Technology, 2015.

[36] Liberatore, M., and Levine, B. N. Inferring the Source of Encrypted HTTP Connections. In *Proceedings of the 13th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2006), CCS '06, ACM, pp. 255–263.

[37] Lu, L., Chang, E., and Chan, M. C. Website Fingerprinting and Identification Using Ordered Feature Sequences. In *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings* (2010), pp. 199–214.

[38] Luo, X., Zhou, P., Chan, E. W. W., Lee, W., Chang, R. K. C., and Perdisci, R. HTTPOS: Sealing Information Leaks with Browser-Side Obfuscation of Encrypted Flows. In *In Proc. Network and Distributed Systems Symposium (NDSS). The Internet Society* (2011).

[39] Maone, G. NoScript Security Suite Add-ons for Firefox. `https://addons.mozilla.org/en-US/firefox/addon/noscript/`. (Accessed on 07/28/2017).

[40] Meier, Bill. Ethernet capture setup. `https://wiki.wireshark.org/CaptureSetup/Ethernet#Capture_on_the_machine_you.27re_interested_in`. (Accessed on 1/08/2017).

[41] Miller, B. HTTPS Vulnerability to Fine Grain Traffic Analysis. technical report, berkeley University, 2012.

[42] MozillaZine. About:config. `http://kb.mozillazine.org/About:config`. (Accessed on 1/08/2017).

[43] Nithyanand, R., Cai, X., and Johnson, R. Glove: A Bespoke Website Fingerprinting Defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society* (New York, NY, USA, 2014), WPES '14, ACM, pp. 131–134.

[44] OpenNet Initiative. Country Profiles. `https://opennet.net/country-profiles`. (Accessed on 06/05/2015).

[45] Panchenko, A., Lanze, F., Pennekamp, J., Engel, T., Zinnen, A., Henze, M., and Wehrle, K. Website Fingerprinting at Internet Scale. In *NDSS* (2016).

[46] Panchenko, A., Niessen, L., Zinnen, A., and Engel, T. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society* (New York, NY, USA, 2011), WPES '11, ACM, pp. 103–114.

[47] Panchenko, A., and Pimenidis, L. Towards Practical Attacker Classification for Risk Analysis in Anonymous Communication. In *Proceedings of the 10th IFIP*

TC-6 TC-11 International Conference on Communications and Multimedia Security (Berlin, Heidelberg, 2006), CMS'06, Springer-Verlag, pp. 240–251.

[48] Perry, M. A Critique of Website Traffic Fingerprinting Attacks. `https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks`. (Accessed on 06/07/2014).

[49] Perry, M. Experimental Defense for Website Traffic Fingerprinting. `https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting`. (Accessed on 06/07/2014).

[50] Perry, M., Clark, E., Murdoch, S., and Koppen, G. The Design and Implementation of the Tor Browser. `https://www.torproject.org/eff/tor-legal-faq.html.en`, April 2015. (Accessed on 06/06/2015).

[51] Reporters without Borders. Enemies of The Internet, Special Edition Surveillance. `http://surveillance.rsf.org/en/`. (Accessed on 06/06/2015).

[52] Ruiz-MartíNez, A. Review: A Survey on Solutions and Main Free Tools for Privacy Enhancing Web Communications. *J. Netw. Comput. Appl. 35*, 5 (Sept. 2012), 1473–1492.

[53] Selenium Project. Selenium Documentation. `http://www.seleniumhq.org/docs/`. (Accessed on 12/06/2017).

[54] Sun, Q., Simon, D. R., Wang, Y.-M., Russell, W., Padmanabhan, V. N., and Qiu, L. Statistical Identification of Encrypted Web Browsing Traffic. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2002), SP '02, IEEE Computer Society, pp. 19–.

[55] Thus, A. Ethical Tor Research: Guidelines. https://blog.torproject.org/blog/ethical-tor-research-guidelines, 11 15. (Accessed on 06/12/2016).

[56] Tor Project. API. https://stem.torproject.org/api.html. (Accessed on 1/08/2017).

[57] Tor Project. How to Torify. https://trac.torproject.org/projects/tor/wiki/doc/TorifyHOWTO. (Accessed on 07/28/2017).

[58] Tor Project. Legal FAQ for Tor Relay Operators. https://www.torproject.org/eff/tor-legal-faq.html.en. (Accessed on 06/07/2016).

[59] Tor Project. Tor browser. https://www.torproject.org/projects/torbrowser.html.en. (Accessed on 07/28/2017).

[60] Tor Project. Tor FAQ. https://www.torproject.org/docs/faq.html.en#TBBJavaScriptEnabled. (Accessed on 06/06/2015).

[61] Tor Project. Tor Project: Anonymity Online. https://www.torproject.org/. (Accessed on 06/05/2015).

[62] Tor Project. Users Tor Metrics. https://metrics.torproject.org/userstats-relay-country.html. (Accessed on 06/04/2017).

[63] Wagner, D., and Schneier, B. Analysis of the SSL 3.0 Protocol. In *Proceedings of the 2nd Conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2* (1996), WOEC'96, pp. 4–4.

[64] Wang, T., Cai, X., Nithyanand, R., Johnson, R., and Goldberg, I. Effective Attacks and Provable Defenses for Website Fingerprinting. In *Proceedings of*

the 23rd USENIX Conference on Security Symposium (Berkeley, CA, USA, 2014), SEC'14, USENIX Association, pp. 143–157.

[65] Wang, T., and Goldberg, I. Improved Website Fingerprinting on Tor. In *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society* (New York, NY, USA, 2013), WPES '13, ACM, pp. 201–212.

[66] Wang, T., and Goldberg, I. Comparing Website Fingerprinting Attacks and Defenses. Tech. rep., Technical Report 2013-30, CACR, 2013. http://cacr. uwaterloo. ca/techreports/2013/cacr2013-30. pdf, 2014.

[67] Wang, T., and Goldberg, I. Walkie-Talkie: An Effective and Efficient Defense against Website Fingerprinting. Tech. rep., Technical Report 2015-09, CACR, 2015. http://cacr. uwaterloo. ca/techreports/2015/cacr2015-09. pdf, 2015.

[68] Wang, X., Luo, J., Yang, M., and Ling, Z. A Potential HTTP-based Application-level Attack Against Tor. *Future Gener. Comput. Syst. 27*, 1 (Jan. 2011), 67–77.

[69] Wireshark. Dumpcap Man Page. `https://www.wireshark.org/docs/man-pages/dumpcap.html`. (Accessed on 1/08/2017).

[70] Wladimir, P. Adblock Plus Add-ons for Firefox. `https://addons.mozilla.org/en-US/firefox/addon/adblock-plus/`. (Accessed on 06/07/2016).

[71] Wright, C. V., Coull, S. E., and Monrose, F. Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis. In *Proceedings of the 16th Network and Distributed Security Symposium* (2009), IEEE, pp. 237–250.

[72] Wu, Peter. Capture Filters. `https://wiki.wireshark.org/CaptureFilters`. (Accessed on 1/08/2017).