University College London

Dept. Computer Science

Automatic Feature Selection for Website Fingerprinting

Axel Goetz

BSc. Computer Science

Supervisors:

Dr. George Danezis

Jamie Hayes

March 17, 2017

Abstract

Contents

1 Introduction

1.1 The Problem

The internet has become an essential tool for communication for a majority of the population. But privacy has always remained a major concern, which is why nowadays most web content providers are slowly moving away from HTTP to HTTPS. For instance, at the time of writing, around 86% of Googles traffic is encrypted. This is a significant improvement compared to 2014 where only 50% was sent over HTTPS [7]. However, this encryption only obscures the content of the web page and does not hide what websites you are visiting or in general who you might be communicating with.

Hence, an Internet Service Provider (ISP) can easily obtain a lot of information about a person. This is an especially large concern for people living in oppressive regimes since it allows a government to easily spy on its people and censor whatever websites they would like. To circumvent these issues, several anonymization techniques have been developed. These systems obscure both the content and meta-data, allowing a user to anonymously browse the web. One of the most popular low-latency anonymization networks is called Tor, which relies on a concept called Onion Routing [24].

The list of known attacks against Tor is at the time of writing very limited and most of them rely on very unlikely scenarios such as having access to specific parts of the network *(both entry and exit nodes)* [24]. However, for this report we will make a more reasonable assumption that an attacker is a *local eavesdropper*. By this we mean that the entity only has access to the traffic between the sender and the first anonymization node, like ISPs.

One of the most successful attacks that satisfies these conditions is known as *website fingerprinting* (WFP). It relies on the fact that Tor does not significantly alter the shape of the network traffic [9]. Hence, the attack infers information about the content by analysing the raw traffic. For instance by analysing the packet sizes, the amount of packets and the direction of the traffic, we might be able to deduce which web pages certain users are visiting. Initially, Tor was considered to be secure against this threat but around 2011, some techniques such as the *support vector classifier* (SVC) used by Panchenko et al. started to get recognition rates higher than 50% [17].

However, one of the main problems with majority of the attacks proposed in the research literature, is that they rely on a laborious, time-consuming manual feature engineering process. The reason why is because most primitive machine learning techniques are only able to process fixed-length vectors as its input. Therefore features need to be picked based on intuition and trial and error processes that reveal the supposedly most informative features.

Thus the goal of this paper is to investigate the use of novel deep-learning techniques to automatically extract a fixed-length vector representation from a traffic trace that represents loading a web page. Next, we aim to use these features in existing attacks to see if our model successfully identified the appropriate features.

1.2 Aims and Goals

We can subdivide the project up into different aims, each with their own goals:

1. **Aim:** Critically review the effectiveness of current website fingerprinting attacks.
   Goals:
   - Analyse various models that are currently used in fingerprinting attacks.
   - Examine how would a small percentage of false positives impacts a potential attack.
   - Analyse how the rapid changing nature of some web pages would impact the attack.
   - Review if there are any assumptions that are being made that could impact the effectiveness of an attack.
2. **Aim:** Automatically generate features from a trace that represents loading a webpage.
   Goals:
   - Critically compare various different feature generation techniques such as stacked authoencoders, RNN sequence-to-sequence and bidirectional RNN encoder decoder models.
   - Identify a dataset that is large enough to train a deep-learning model.
   - Compare several software libraries to perform fast numerical computation such as Tensorflow, Torch, Theano and Keras.
   - Implement the most appropriate feature generation model in one of the previously mentioned software libraries.
3. **Aim:** Train existing models with the automatically generated features and test their performance compared to hand-picked ones.
   Goals:
   - Identify five different models that have previously been successful in various website fingerprinting attacks and implement those models.
   - Extract the same hand-picked features from our dataset as mentioned in the respective papers.
   - Investigate an appropriate technique for evaluating the results of different models.
   - Compare the hand-picked features compared to the automatically generated ones for different models. In addition, we also want to investigate their effectiveness in different threat models. For instance if an adversary wants to identify which specific web pages a user is visiting *(multi-class classification)* or if the adversary just wants to know whether the user visits a web page from a monitored set of web pages *(binary classification)*.

1.3 Project Overview

As previously mentioned, the general project can be split up into three different aims. Hence, we also approached it in three stages:

- First, we examine different existing website fingerprinting models to gain a deeper understanding of the concept.
- Next, we perform more research to contrast different automatic feature selection models and implement the most appropriate model.
- Finally, we compare the effectiveness of hand-picked features with the automatically generated ones by training them on different existing website fingerprinting models.

1.4 Report Structure

The general report has a very simple infrastructure. In the following section we further explore several concepts that are necessary to understand the basics of website fingerprinting and the specific attack that we will be performing in our experiment. Next, we identify the threat model and design an attack for that threat model that automatically generates features. Finally, we explore several methods of evaluating the performance of different models with different features and contrast the hand-picked features with the automatically generated ones. ../report.bib

2 Background Information, Related Work and Research

In the following chapter, we further explore the motivation for undertaking the project, analyse the current state of the project domain and outline the research that forms the basis for the rest of the report.

2.1 The Problem

As previously mentioned, the goal of this project is to automate the feature selection process for a website fingerprinting attack. By this we mean that given a specific trace, our model should be able to produce a fixed-length vector that is a close representation of the respective trace. However, before we delve into the details of the attack, we first need to gain a greater understanding of some concepts such as onion routing, website fingerprinting in general and deep learning.

2.1.1 Onion Routing

To preserve privacy, we do not only need to obscure the content of a webpage but also hide who is taking to whom [6]. Tor achieves both of these by making use of a technique, called *onion routing*, which is a very simple protocol that can be divided up into three phases: connection setup, data movement and connection tear-down [6]. We show how it works by taking a simple example of Alice trying to communicate with Bob.

1. **Connection Setup:**
    - Alice's Tor client obtains a list of Tor nodes from a directory server.
    - Then Alice picks three random Tor nodes and labels them *one, two* and *three*.
    - Alice communicates with the first node and shares a symmetric key.

- Next, Alice sends messages to the first node, which are then forwarded to the second node to share another symmetric key to the second node.
- Finally, Alice continues sending messages to the first node, which are forwarded to the second and finally to the third node to share the final symmetric key. What is important here is that we use a secure key-sharing algorithm such that only Alice and the respective node know the keys. Additionally, since all of the traffic is forwarded from the first node, the second and the third nodes do not know the identity of Alice.

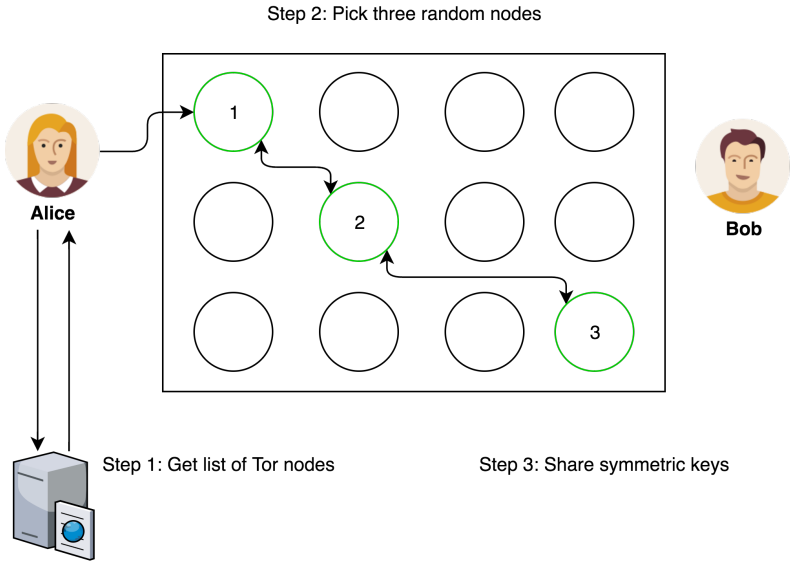

Figure 2.1: An example of a connection setup for the onion routing protocol.

2. **Data Movement:**

- Before Alice can send any data, it first needs to encrypt it in different layers. By this we mean that first it encrypts the data *(with Bob's address)* using the shared key from the third node. Next, it encrypts that data again *(with the address of the third node)* using the key from the second node. Finally, as expected, it encrypts the data a final time *(with the address of the second node)* using the shared key from the first node.
- Now Alice is ready to send the data to the first node.
- Once the first node received the data, it decrypts it using the shared key. This reveals the address to the second node. The key is here that the first node cannot see the data nor where it is going, since that is still encrypted.
- Next, the first node forwards the data that it just unencrypted to the second node. Again, this node decrypts the data, revealing the address to the third node but now it doesn't know what the data is, where the final destination is or where it originally came from.
- Lastly, the second node forwards the data to the third node. After encryption, this final node can see the data and where it is going but it does not know where it came from. So it forwards the data to Bob and not a single party should be able to know the data, the final destination and where it originally came from except for Alice and Bob.

Now we know why the protocol is called onion routing because it encrypts the data in multiple layers and at every node, one of the layers of the onion is peeled off [20]. The key is that none of the nodes know the complete path that has been taken.

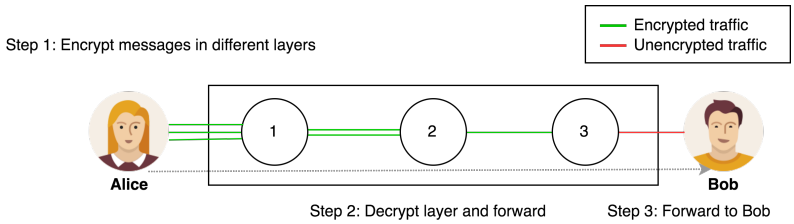

Figure 2.2: Sending a message with the onion routing protocol.

3. **Connection Tear-down:** This can be initiated by any of the nodes or the client and the process is very straightforward. Either the client sends a request for a tear-down to the first node to remove any data on the connection (including the shared key), which is then forwarded to the other nodes. Or one of the nodes sends a tear-down message to both the previous node and the next node, which are then forwarded in both directions [6].

Tor generally uses the same circuit for connections that happen within around 10 minutes after creating a circuit. Later requests, however, are given a new circuit [24, 20].

2.2 Related Work

2.2.1 Website Fingerprinting

Website fingerprinting (WF) is the process of attempting to identify which web pages a specific client visits by analysing their traffic traces. Hence, an attacker is considered to be *local* . By this we mean that they can eavesdrop on the traffic between the client and the first Tor node, as shown in figure 2.3. So it can be anyone from a person on the same network to an ISP. The reason as to why the attacker has to be local is because in onion routing systems, it is the only place in the network where you still know the true identity of the client.

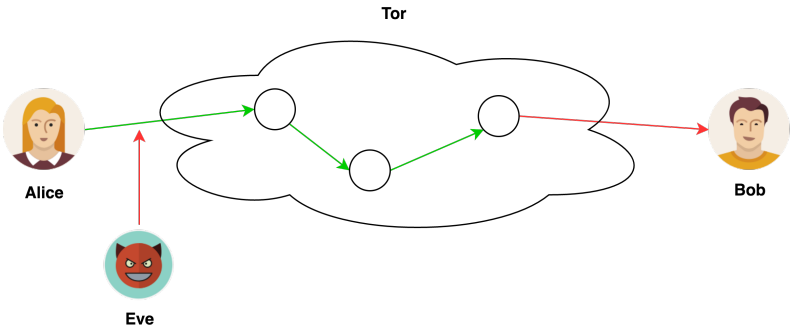

Figure 2.3: Threat model for a simple website fingerprinting attack.

In general, the attack is as follows. The attacker first collects a dataset that contains packet traces for loading several web pages. In practice, these web pages are the ones that an attacker is interested in monitoring. From those traces, the attacker extracts a fixed number of features, the so-called *fingerprint*. Next, it uses those fingerprints to train some

sort of model *(often a machine learning model)* to find patterns in the data. The attacker observes packet traces, generated by the client, and uses the model to classify which web pages the user is visiting. Therefore, an attack is essentially a classification problem, where you try to map packet traces into web pages.

We should denote that throughout this report, we will be using the word "web pages" rather than "websites". The reasoning behind this is that a website often consists of multiple HTML documents, which can result in significantly different traffic traces, depending on which document you load.

The term *"website fingerprinting"* was first mentioned by A. Hintz who performed a simple traffic analysis on Safeweb, an encrypting web proxy that tried to protect user's privacy [11]. Although the attack was very simple, it and other earlier works show the possibility that encrypted channels might still leak information about the URL [11, 25]. Later, in 2009, the first fingerprinting attack against Tor was executed by Herrmann et al. using a *Multinomial Naive Bayes* model. They managed to get a relatively high accuracy for several protocols, such as *OpenSSL* and *OpenVPN*, on a small set of 775 web pages. However on more specialised anonymization networks, the model only achieved a 2.96% for Tor, which they said was caused by a suboptimal configuration [10].

Around the same period Y. Shi et al. performed an attack that specifically focused on anonymity systems such as Tor [22]. Using a cosine similarity, they achieved around 50% accuracy on an even smaller dataset of only 20 web pages [22]. The first real threat however, was by A. Panchenko et al. who used a *Support Vector Classifier* (SVC) on the same dataset of 775 web pages as Herrmann et al. and got a 54.61% success rate [10, 17].

All of the previously mentioned research, except the one done by Panchenko et al. have only considered a *closed-world scenario*. A closed-world setting means that all of the web pages are known in advance [17]. For instance, when Herrmann et al. trained their model on a dataset of 775 web pages, they made the assumption that a client could only visit one of those web pages and none other. In an *open-world scenario*, the attacker does not know in advance which URLs the victim might visit. The most prominent example of this is when the authorities want to monitor which people try to access a set of censored sites [17]. In order to achieve this, the models need to be trained on both *monitored* and *unmonited web pages*.

Wang et al. later conducted an attack on Tor using a large open-world dataset [27]. Using a novel *k-Nearest Neighbor* (k-NN) classifier with *weight adjustment* on a very large feature set *(3736 features)*. In addition to getting around 90% accuracy, they also significantly reduced the time needed for training [27].

Using a completely different approach, Hayes et al. extract fingerprints using a *random forests* [9]. This novel technique involved storing a *fingerprint* as a set of leaves within that forest. Next, they simply use the *hamming distance* as a distance metric to find the k-nearest neighbors. If all the labels within those k instances are the same, the model classifies the new instance as the previously mentioned label. The interesting aspect is that changing the value of k allows them to vary the number of *true positives* (TP) and *false positives* (FP) [9].

Finally, Panchenko et al. improved upon their previous attack to create one current state-of-the-art methods. They tested their approach on several datasets, including the one used by Wang et al. on their k-NN attack, where they got around 92% accuracy. In an open-world scenario, on the other hand, they got up to 96% accuracy.

2.2.2 Defenses

Not only has there been research regarding different attacks but there are also various works that describe possible defenses. First of all, Tor already implements *padding*, which means

that all packets are padded to a fixed-sized cell of 512 bytes. Next, in response to the first attack by Panchenko et al., Tor also supported randomized ordering of HTTP pipelines [17, 9, 19]. Finally, on top of these defenses, fingerprinting on Tor is made more difficult by all of the background noise present. This is due to the fact that Tor also sends packets for circuit construction or just *SENDME's*, which ensure flow control [16]. Although Wang et al. proposed a probabilistic method to remove these [26], they might still make the classification process slightly more difficult.

Lua et al. designed an application-level defense, that was able to successfully defends against a number of classifiers by modifying packet size, timing of packets, web object size, and flow size [19]. This is achieved by splitting individual HTTP requests into multiple partial requests, using extra HTTP traffic as a cover and making use of HTTP pipelining [2]. Although this has been a relatively effective technique to obfuscate traffic, several attacks have proven that this defense only still does not suffice [2, 27].

BuFLO, on the other hand, is a *simulatable* defense [27], designed by Dyer et al. that performed packet padding and sending data at a constant rate in both directions [5]. The disadvantage of this method is the high overhead required in order to keep the constant data rate. Some extensions have been described that try to minimize this overhead such as Nithyanand's work that uses existing knowledge of a website traces to maintain a high level of security [13].

More recently, Cherubin et al. developed the first website fingerprinting defense on the server side [3]. This can be particularly interesting for *Tor hidden services* that want to provide, all of their users, the privacy that they require. The attack uses a technique, called *ALPaCA*, which pads the content of a web page and creates new content to conceal specific features on a network level [3].

Finally, there are also some other techniques such as *decoy pages* and *traffic morphing* [29, 17]. Decoy pages or *camouflage* is a very simple technique that involves loading another web page whenever a web page is requested. This process provides more background noise that makes fingerprinting more difficult [17]. Whilst traffic morphing is a slightly more complex technique that changes certain features in the traffic in order to make it appear as if another page is loaded [29].

2.2.3 Critical Analysis

- Tor version - Closed world - Well defined packet traces (It is assumed that the attacker knows where the packet trace of a singlenpage load starts and ends. If the client takes much longer to load the next page after the current one is loaded, this assumption can be justified.) - No other activity (multi-tab browsing)

(Website fingerprinting at Internet scale proves that the attack does not scale in realistic settings)

TODO: How it performs against defenses TODO: Regularisation TODO: Tor hidden services

Bibliography

[1] Kota Abe and Shigeki Goto. "Fingerprinting Attack on Tor Anonymity using Deep Learning". In: *Proceedings of the APAN Research Workshop 2016* ().

[2] Xiang Cai et al. "Touching from a Distance: Website Fingerprinting Attacks and Defenses". In: *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12* (2012). DOI: 10.1145/2382196.2382260. URL: http://pub.cs.sunysb.edu/~rob/papers/fp.pdf.

[3] Giovanni Cherubin, Jamie Hayes, and Marc Juarez. "Website Fingerprinting Defenses at the Application Layer". In: *Proceedings on Privacy Enhancing Technologies* 2 (2017), pp. 165–182.

[4] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2014). DOI: 10.3115/v1/d14-1179. URL: https://arxiv.org/pdf/1406.1078v3.pdf.

[5] Kevin P Dyer et al. "Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail". In: *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE. 2012, pp. 332–346.

[6] David Goldschlag, Michael Reed, and Paul Syverson. "Onion routing". In: *Communications of the ACM* 42.2 (1999), pp. 39–41.

[7] Google. "Google Transparency Report". In: (Mar. 2017).

[8] Xiaodan Gu, Ming Yang, and Junzhou Luo. "A novel Website Fingerprinting attack against multi-tab browsing behavior". In: *2015 IEEE 19th International Conference on Computer Supported Cooperative Work in Design (CSCWD)* (2015). DOI: 10.1109/cscwd.2015.7230964.

[9] Jamie Hayes and George Danezis. *k-fingerprinting: A Robust Scalable Website Fingerprinting Technique*. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/hayes.

[10] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. "Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naive bayes classifier". In: *Proceedings of the 2009 ACM workshop on Cloud computing security*. ACM. 2009, pp. 31–42.

[11] Andrew Hintz. "Fingerprinting websites using traffic analysis". In: *International Workshop on Privacy Enhancing Technologies*. Springer. 2002, pp. 171–178.

[12] Marc Juarez et al. "A Critical Evaluation of Website Fingerprinting Attacks". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security - CCS '14* (2014). DOI: 10.1145/2660267.2660368. URL: http://www1.icsi.berkeley.edu/~sadia/papers/ccs-webfp-final.pdf.

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

[14] Christopher Olah. *Understanding LSTM Networks*. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[15] Barak Oshri and Nishith Khandwala. "There and Back Again: Autoencoders for Textual Reconstruction". In: (). URL: http://cs224d.stanford.edu/reports/OshriBarak.pdf.

[16] Andriy Panchenko et al. "Website Fingerprinting at Internet Scale". In: (). URL: https://www.comsys.rwth-aachen.de/fileadmin/papers/2016/2016-panchenko-ndss-fingerprinting.pdf.

[17] Andriy Panchenko et al. *Website fingerprinting in onion routing based anonymization networks*. 2011. DOI: 10.1145/2046556.2046570. URL: https://www.freehaven.net/anonbib/cache/wpes11-panchenko.pdf.

[18] Mike Perry. *A Critique of Website Traffic Fingerprinting Attacks*. URL: https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks.

[19] Mike Perry. "Experimental defense for website traffic fingerprinting". In: *Tor project Blog* (2011). URL: https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting.

[20] The Tor Project. "Tor Overview". In: (). URL: https://www.torproject.org/about/overview.html.en.

[21] Vera Rimmer. "Deep Learning Website Fingerprinting Features". In: (2016). URL: https://www.esat.kuleuven.be/cosic/publications/thesis-280.pdf.

[22] Yi Shi and Kanta Matsuura. "Fingerprinting attack on the tor anonymity system". In: *International Conference on Information and Communications Security*. Springer. 2009, pp. 425–438.

[23] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: (). URL: https://arxiv.org/abs/1409.3215.

[24] *The Tor Project*. URL: https://www.torproject.org/docs/faq.

[25] David Wagner, Bruce Schneier, et al. "Analysis of the SSL 3.0 protocol". In: *The Second USENIX Workshop on Electronic Commerce Proceedings*. Vol. 1. 1. 1996, pp. 29–40.

[26] Tao Wang and Ian Goldberg. "Improved website fingerprinting on Tor". In: *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society - WPES '13* (2013). DOI: 10.1145/2517840.2517851. URL: https://www.freehaven.net/anonbib/cache/wpes13-fingerprinting.pdf.

[27] Tao Wang et al. "Effective Attacks and Provable Defenses for Website Fingerprinting". In: *USENIX Security Symposium* 23 (Aug. 2014). URL: https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-wang-tao.pdf.

[28] Zhanyi Wang. "The Applications of Deep Learning on Traffic Identification". In: (). URL: https://www.blackhat.com/docs/us-15/materials/us-15-Wang-The-Applications-Of-Deep-Learning-On-Traffic-Identification-wp.pdf.

[29] Charles V Wright, Scott E Coull, and Fabian Monrose. "Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis." In: *NDSS*. Vol. 9. 2009.