UNIVERSITEIT VAN AMSTERDAM

System and Network Engineering

Website fingerprinting attacks against Tor Browser Bundle: a comparison between HTTP/1.1 and HTTP/2

Research Project #1

Authors:
T.T.N. Marks BSc.
K.C.N. Halvemaan BSc.

February 12, 2017

Abstract

Users of routing networks such as *The Second-Generation Onion Router* (Tor) expect a high level of anonymity, however, website fingerprinting of Tor traffic can be done with a high accuracy based on metadata of the encrypted data in the *Transmission Control Protocol / Internet Protocol* (TCP/IP) stream. An eavesdropper listening in on the traffic between the client and the guard node can get an accurate assumption of what website a user has visited by comparing the current stream to a labelled stream from a database that has been collected beforehand. This paper gives a comparison of website fingerprinting attacks between a *Tor Browser Bundle* (TBB) using just *Hypertext Transfer Protocol 1.1* (HTTP/1.1) and a TBB using *Hypertext Transfer Protocol 2* (HTTP/2). HTTP/2 has not been enabled by default in the TBB because the code has not been audited and the security implications of enabling it have not been examined. This paper contributes to this study of the implications of using HTTP/2 in TBB. In closed-world experiments an average accuracy of 88.036% ($s = 2.0164\%$) was achieved for HTTP/1.1 and an average accuracy of 86.4585% ($s = 3.0871\%$) for HTTP/2. When training with HTTP/1.1 and testing with HTTP/2 an average accuracy of 64.687% ($s = 6.6631\%$) was achieved. When training with HTTP/2 and testing with HTTP/1.1 an average accuracy of 54.667% ($s = 3.5286\%$) was achieved. The accuracy of an attacker's website fingerprinting attack will suffer between 20% and 40% when doing a website fingerprinting attack when the *Hypertext Transfer Protocol* (HTTP) version differs in his model from the one used by its target.

Contents

Chapter 1

Introduction

Tor was introduced in 2004 as a circuit-based low-latency anonymous communication service [Dingledine et al., 2004]. Thus users of routing networks such as Tor expect a high level of anonymity. This is reflected on the Tor project website: "Tor's users employ this network by connecting through a series of virtual tunnels rather than making a direct connection, thus allowing both organisations and individuals to share information over public networks without compromising their privacy"[1].

Despite providing mechanisms to protect its users against network level attacks, Tor is still vulnerable to website[2] fingerprinting attacks as indicated by earlier research [Panchenko et al., 2011][Wang and Goldberg, 2013]. These passive attacks can occur when someone can eavesdrop on the traffic from the client to the guard node. All of the previous research done on fingerprinting attacks has been based on traffic using HTTP/1.1. At the time of writing, the default TBB does not have HTTP/2 enabled yet by default as its implementation has not been audited by the Tor project[3].

There are significant differences between the two versions of the HTTP [Morla, 2016]. Research on website fingerprinting attacks with HTTP/2 will benefit the community at large since HTTP/2 will be probably be enabled by default in the TBB in the future, as HTTP/2 contains numerous improvements over HTTP/1.1 and is already enabled in all major browsers. Additionally, this research might provide more insight into the security implications of using HTTP/2.

This paper will attempt to reproduce the research done by Wang and Goldberg [2013] and focus on the differences between HTTP/1.1 and HTTP/2 on the possibilities of website fingerprinting attacks. The following research questions have been formulated:

1. Can a website fingerprinting attack be done on TBB when visiting websites via HTTP/2?
2. What are the differences in fingerprinting attacks on a TBB enabled with just HTTP/1.1 and a TBB enabled with HTTP/2?

The data has been generated by the authors by visiting regular websites from the Alexa top websites in order to prevent both privacy issues, and illegal content being visited on the web[4]. The research

[1] `https://www.torproject.org/about/overview.html.en`

[2] A 'website' in this paper will be synonymous to 'web page'.

[3] `https://trac.torproject.org/projects/tor/ticket/14952`

[4] Note that the list of URLs contains adult content, but this should form no problem from a legal perspective.

will focus only on the client side (the TBB). Only traffic generated from TBB instances under control of the authors is used[5].

An overview of the related work on website fingerprinting as well as a brief explanation on the workings of Tor is given in Section 2. The experimental setup used to get the data for the website fingerprints, as well as the method of converting the data and training the SVM can be found in Section 3. The results of the experiments are shown in Section 4. A conclusion is given in Section 5 and a discussion in Section 6.

[5]These instances only access the Tor network and will not be acting as relays.

Chapter 2

Related Work

A brief overview of the workings of Tor is described in Section 2.1. The differences relevant for this research between HTTP/1.1 and HTTP/2 are explained in Section 2.2. An introduction in website fingerprinting is given in Section 2.3 as well as related research on website fingerprinting of Tor traffic.

2.1 Tor

Tor is a protocol designed for traffic anonymisation. Instead of a client connecting directly to a server its traffic is tunnelled through a number of Tor relays in order to hide the *Internet Protocol address* (IP address) of the client to the endpoint. The Tor nodes are run by volunteers from across the globe. Tor can be used by regular applications as a proxy, or prepackaged with a web browser, the TBB. In addition to the anonymisation mentioned earlier, the TBB also applies techniques to combat browser fingerprinting.

When connecting to Tor, a client first gets a list of Tor nodes from a Tor directory server (which also may also act as a Tor node itself). Using this list of nodes Tor the client will create a circuit of nodes, usually existing of three nodes with each a specific function based on its position in the circuit: a guard node, a relay node and an exit node. These are respectively, the first, second and third node. Not every node in the directory listing is fit to be any of the three types listed. Nodes with high throughput and uptime are used as guard nodes and exit nodes. For each few web pages that are loaded in a pre-determined timeframe the TBB will use the same circuit. For a certain number of circuits, the guard node will stay the same and only the relay and exit node are changed. Hosting an exit node adds a legal burden to the host, since for the outside world all the traffic coming from this node seems to be originating from the host's IP address. A visual representation of the process can be seen in Figure 2.1.

The client will create multiple streams which correspond to TCP/IP connections that go through a specific Tor circuit. Tor encrypts the traffic with *Transport Layer Security* (TLS) and transfers its data in units of 512 bytes, called Tor cells. In the case of the TBB, a stream is attached to a circuit for each HTTP request.

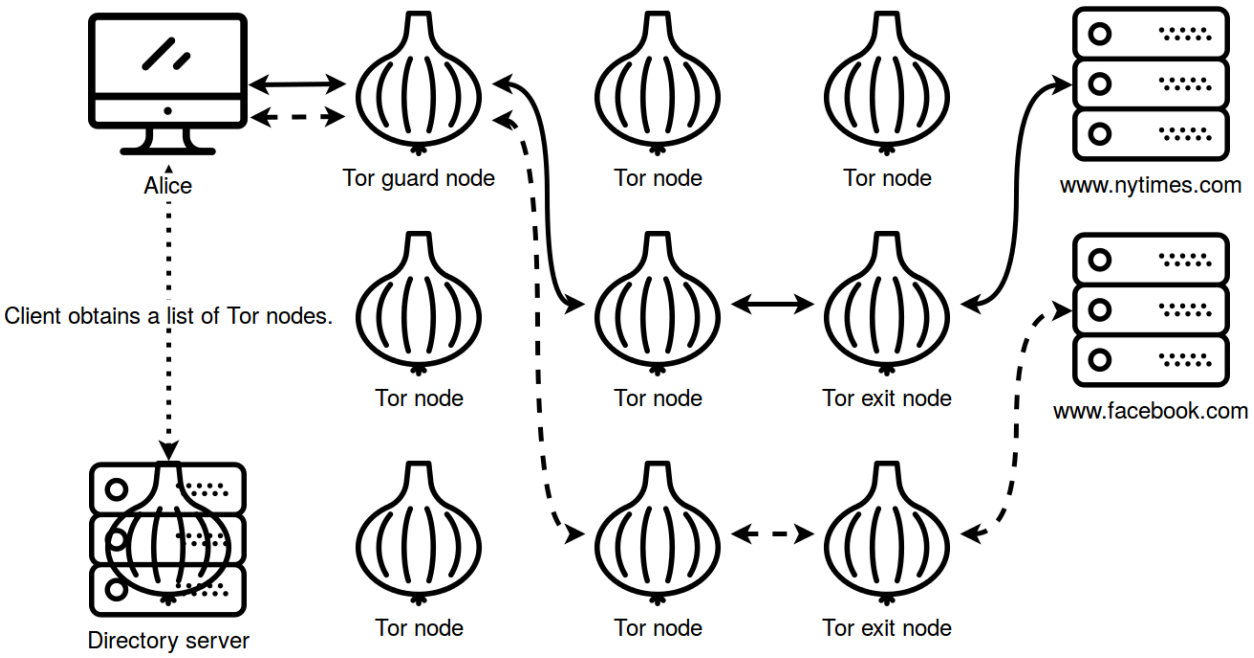

Figure 2.1: Step 1 (dotted): The client connects to the Tor directory servers to retrieve a list of Tor nodes. Step 2 (solid): The client connects through Tor to `www.nytimes.com`. Step 3 (dashed): The client connects through a different circuit to `www.facebook.com`.

2.2 HTTP/2

As mentioned in Section 1 all previous research on Tor website fingerprinting was done using only HTTP/1.1. There are significant differences in HTTP/2 compared to HTTP/1.1. Some of the differences that are relevant to this research are explained in the following section.

The first important difference is that when HTTP/2 is used as a transport for web browsing in the browser TLS is required, even though TLS is not required according to the spec [Belshe et al., 2015]. However, the browser vendors have decided to make this a hard requirement[WG, 2017], making it a *de facto* requirement as most HTTP/2 traffic is web browser based. The most profound change of HTTP/2 is that the protocol now supports multiplexing. This enables multiple different requests to make use of a single HTTP/2 connection concurrently [Belshe et al., 2015]. This benefits web browsers as they no longer have to setup multiple TCP/IP connections [Ouders, 2008]. Furthermore, the multiplexing solves the *Head-of-line blocking* (HOL) problem that can occur when using the HTTP/1.1 pipelining feature to further speed-up page load times [Belshe et al., 2015]. The last relevant change of HTTP/2 enables clients to signal the server in what order it would like to receive requested files [Belshe et al., 2015]. The client can do this by prioritising certain streams within the multiplexed connection or marking them as dependencies of other streams.

The authors think that the multiplexing and prioritisation specifically could have implications for the packet streams that are used in website fingerprinting and thus warrant further investigation.

2.3 Website fingerprinting

In Figure 2.2 a website fingerprinting attack is illustrated[1]. An attacker would start by capturing TCP/IP packets while accessing websites via the Tor network, to derive the so-called fingerprint of a website. The attacker will then be listening in on the encrypted connection between the client and the guard node, and compares the fingerprints it had made earlier with the stream he is currently seeing. When the fingerprint of the current TCP/IP stream matches with one in his database, he will be able to identify which website the client had accessed.

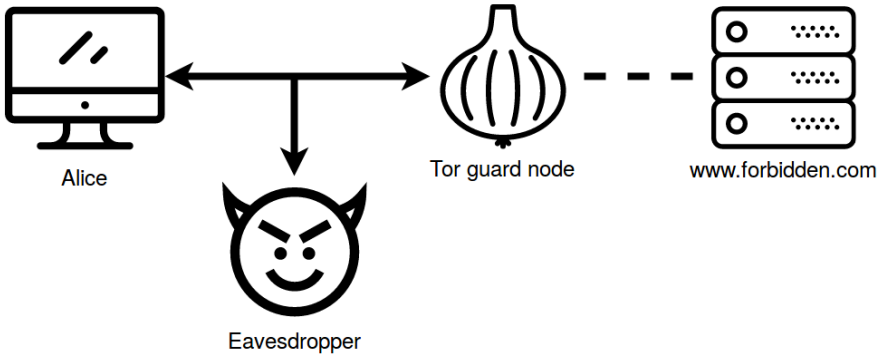

Figure 2.2: An attacker is eavesdropping on the communication between the client and the guard node and can infer based on the metadata of the TCP/IP stream which website the client is visiting.

Liberatore and Levine [2006] published two methods for fingerprinting encrypted HTTP traffic using a Naïve Bayes classifier by looking at the length and direction of TCP/IP packets while discarding the order. This research was then further extended to the TBB by Herrmann et al. [2009], with improved accuracy under comparable conditions by using a Multinomial Naïve Bayes classifier. Improvements on identifying traffic were made on the previous research by using a SVM instead of the Naïve Bayes classifier [Panchenko et al., 2011]. In addition, the paper of Panchenko et al. gives an extensively detailed description of the setup to repeatedly request pages, as well as the features used for the classification, which was incomplete or missing in earlier research. A comparison of various defences, some of which have been implemented in the Tor browser, is given in Cai et al. [2012]. A review of earlier methods was given in Wang and Goldberg [2013] as well as an improvement of accuracy while modelling a more realistic approach than was done previously[2]. This consisted of using multiple circuits during testing to get various localised versions of web pages and using a Tor controller[3] to gain full control over circuit construction.

All the previous research was done using HTTP/1.1. With the comparison between HTTP/1.1 and HTTP/2 in this paper the authors hope to pave the way for further research into the security implications of enabling HTTP/2 in the TBB. In addition, it might allow for research into the effect of multiplexing and prioritising as part of HTTP/2 and possible applications of these features as defences against website fingerprinting.

[1] Figure 2.2 has part of the Tor network left out for clarity purposes (indicated by the dashes between the guard node and the web server), the whole view can be seen in Figure 2.1.

[2] Wang and Goldberg [2013] also have provided access to their source and those of related research on `https://crysp.uwaterloo.ca/software/webfingerprint/`.

[3] `https://svn.torproject.org/svn/blossom/trunk/TorCtl.py`

Chapter 3

Method

In Section 3.1 the method is explained of the gathering of the list of website URLs on which the fingerprinting has been performed. The various software tools that were used to configure and use the TBB are discussed in Section 3.2.1. The automation of retrieving the websites using Selenium is explained in Section 3.2.2. The criteria and method for the validation of the captured data are laid out in Section 3.3. The extraction of features from the gathered data is explained in Section 3.4 as well as the application of the SVM on these features in Section 3.5.

3.1 Website URLs

The sample of websites on which the fingerprinting experiments have been run on is based on the Alexa top 1,000,000 list[1]. The exact version that was used for selecting the sample of the experiments was retrieved on 2017-01-14. The top 5,000 were selected as a basis to start tests to see which websites supported HTTP/2. These tests were done using curl (7.52.1) compiled with HTTP/2 support and sending HTTP/2 HEAD requests to the *HTTP over TLS* (HTTPS) version of the URLs in the list. Only HTTPS URLs were tested because of TBB only supporting HTTP/2 when also using TLS. Of the top 5,000 websites only 1,110 websites responded with HTTP/2 and TLS, which is 22.2%. This is only a small increase compared to similar research done in 2016 has shown that 18% of the websites in the top 5,000 of the Alexa ranking had HTTP/2 enabled [Morla, 2016].

From the list of 1,100 HTTP/2 capable websites variations of the Google search domain (e.g. google.nl, google.ca, google.de, etc.) were removed. This was done to limit variations of these domains within the first 100 entries as there were 47 Google domains in the top 100. Just as Wang and Goldberg [2013] it was deemed that fingerprinting each variation was unnecessary, since they all point to a very similar website. Only the google.com was retained in the list since it is the top 1 website and it behaves interestingly: it refers the user to a different domain based on the user's *Internet Protocol* (IP) address. In addition to Google domains two Chinese domains (tmall.com and sohu.com) were removed which often did not load during preliminary testing. The first 130 entries after the removals were used as the sample set for gathering data. This number is comparable to the 100 URLs Wang and Goldberg [2013] used, apart from that the list now only includes

[1] `https://aws.amazon.com/alexa-top-sites/`

websites which support both HTTP/2 and HTTP/1.1. This is important to be able to do a proper comparison between HTTP versions.

3.2 Implementation

The various software packages that have been used are explained in Section 3.2.1 and the implementation of the crawler in discussed in Section 3.2.2.

3.2.1 Software used

The TBB (6.0.8)[2] has been used which includes a modified version of Mozilla Firefox (45.6.0esr) and Tor (0.2.8.11). The version of Mozilla Firefox that the Tor project includes in this bundle did not have HTTP/2 enabled by default yet. However, it is possible to enable it by changing the configuration in the about:config settings page of the browser. The configuration changes are detailed in Appendix A. An effort was made to change as little as possible to the configuration in order to stay as close as possible to the defaults and thus more likely to reflect real world behaviour. An important change that was made was the disabling of caching in the TBB, so it could not influence multiple visits to the same website, and disabling the auto-update settings for the TBB and the browser add-ons. This was done to ensure that updates would not interfere with the data gathering.

Additionally, the Tor controller library stem (1.5.4) was used to get detailed information from Tor, its configuration and to control its behaviour. To automate the browser, e.g. visiting websites in a particular order, the browser automation software Selenium (2.53.6) was used together with a driver that was specifically built for the TBB, tbselenium (0.1). tbselenium also changed the browser configuration in order to facilitate the usage of the webdriver, which is used by Selenium to automate the control of the browser. These tbselenium specific changes can be found in the repository of the library[3].

A Firefox add-on, *Hypertext Transfer Protocol Archive* (HAR) Export Trigger[4]. was used to enable saving of HAR[5] files automatically through a JavaScript call. HAR files contain information about a page visit encoded in *JavaScript Object Notation* (JSON) such as the request and response headers. The headers in particular were of importance to be able to verify if the traffic captures from Tor were correct, because the *packet capture*s (pcaps) themselves were encrypted and thus not readable.

[2] Release notes: `https://blog.torproject.org/blog/tor-browser-608-released`.
[3] `https://github.com/webfp/tor-browser-selenium/blob/v0.1/tbselenium/tbdriver.py`
[4] `http://www.softwareishard.com/blog/har-export-trigger/`
[5] `http://www.softwareishard.com/blog/har-12-spec/`

3.2.2 Implementing the crawler

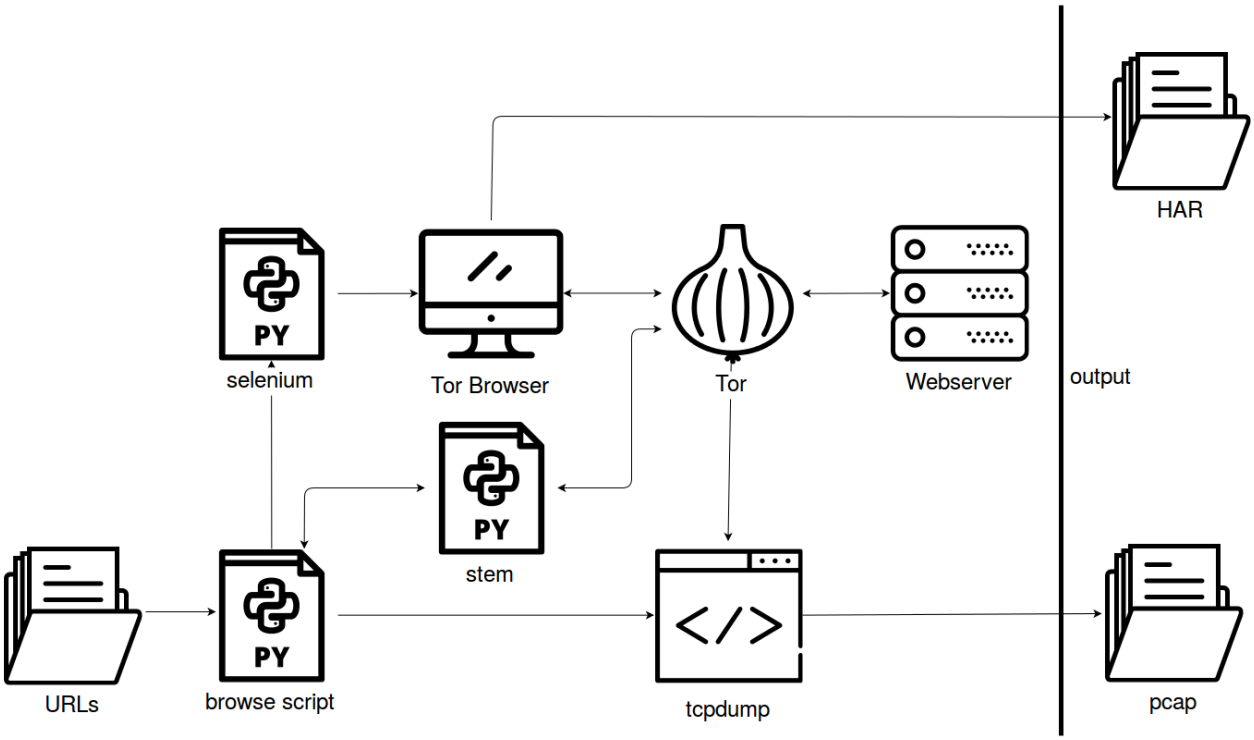

Figure 3.1: An overview of the setup used in the experiments.

A crawler was built with the components mentioned in Section 3.2.1. The crawler was used to retrieve each website in the URL list 40 times as this was considered to be a good sample size for training the machine learning model as indicated in earlier research done by Wang and Goldberg [2013]. A symbolic view of the setup can be seen in Figure 3.1.

During the setup phase the script started the TBB and applied the settings for Selenium and the specified profile to use. During the experiments two different profiles were used, one with and one without HTTP/2 enabled. After starting the TBB with a specific profile the script connects with stem (the Tor controller) to Tor and changes Tor settings to assign circuits and streams manually. Handling the connections of the TBB was necessary to be able to set appropriate filters for tcpdump to capture only traffic to and from the current guard node. This information could be retrieved by explicitly assigning streams to specific circuits, and looking up the IP address and the *Onion Routing port* (ORport) from the guard node of that circuit.

After the setup phase the script starts iterating over a *comma-separated values* (CSV) file that contains the list of websites to be captured. A capture of a website is taken by first retrieving the current guard node IP address and ORport. Next the script starts tcpdump on the host computer with an IP address and port filter corresponding to the guard node IP address and ORport so it only captures Tor traffic. When tcpdump is running, a Selenium command is issued to retrieve an URL from the list. After Selenium reports if the page has been loaded an additional 10 second sleep takes place to allow for *Asynchronous JavaScript and XML* (AJAX) elements to load. After this interval the browser was explicitly signalled to stop loading the page and stop AJAX calls by starting the JavaScript debugger. Finally the packet capture of tcpdump was stopped and an

export was made of the corresponding HAR of the page load visit, to be used later on for validating the contents of the encrypted packet captures. If the website did not load or the script encountered any other issues which could lead to an incorrect or unverifiable capture, the website together with the current iteration gets added to a list to retry at the end of a run.

The Tor network is run by volunteers, and not all nodes have great connectivity. It is common for websites to not load correctly caused by having to go through multiple nodes in the Tor network. In addition, the manual connection management that was done via the Tor controller library took some experimentation to get working, as there was no official documentation on how to do this properly. This was further complicated by a feature called path bias testing. Tor uses this path bias testing to detect if there is a bias in the circuits that are created by Tor. This path biasing feature is still experimental according to the Tor manual, but has been included in the stable builds since 2012. Documentation of this feature was non-existent at the time of the experiments, besides a very brief description in the Tor settings it is not mentioned anywhere else[6]. As a result of the bias testing, nodes in a circuit will start to reject new connections after a configured number of previously handled connections. This combination made manual assignment of Tor streams to circuits challenging.

The values that were used during this research were 8 websites per circuit and switching the guard node after each iteration of the URL list, so after 40 websites. These parameters meant that in general each of the 40 visits to a specific website was done over a different circuit. Thereby being able to train the SVM on multiple localised versions of the web page based on the IP address of the exit node. For websites without localisation, there would be no difference based on the choice of exit node.

3.3 Validating the data

The packet captures that have been collected with the crawler contain little usable information because the content of the packets have been encrypted by Tor. However, in contrast to earlier papers on fingerprinting Tor traffic it is now important to be able to distinguish HTTP/1.1 from HTTP/2 traffic. This is where the HAR that corresponds to specific packet captures of website plus iteration plays an important role.

A Python script was written to iterate over the files in a folder containing pcaps of website visits created by the crawler in order to validate the resulting visits (thus a valid HAR and corresponding pcap must be found). The script does this in two parts, first by applying some pre-processing steps and finally the validation of the HAR content.

The pre-processing starts by checking if for every website and every iteration thereof it can find a pcap file. Next it will delete and rename HAR files to match pcaps. This step was necessary because on a retry the pcap file would be overwritten, but for each retry a HAR file would be added with a suffix of the retry attempt. Thus the latest HAR file and the pcap file would correspond to the latest conclusive iteration of a website, all other HAR files were deleted. A situation where no HAR file was present would arise when a timeout had occurred when retrieving the website. The final pre-processing step thus was to see if every pcap now had a matching HAR, if not it was marked invalid as there is no way to know if the captured traffic was retrieved with the intended HTTP version and contains the correct website.

[6]Related Bug-report: `https://trac.torproject.org/projects/tor/ticket/19288`.

After the pre-processing the actual validation of the HAR was done in order to see if the packet capture satisfies the below predetermined set of validation conditions.

1. The HAR has to start with a HTTP GET request.
2. (Optional) There can be one or multiple redirects.
3. After the possible redirects there has to be a HTTP 200 OK response.
4. The response must have a "text/html" as content-type.
5. If the request was made using HTTP/2, the response should be in HTTP/2.

If these conditions were met the website visit and corresponding packet capture was deemed valid. The HTTP redirects to different domains made it impractical to check if the page that was loaded corresponded to the domain of the initial request. The final condition of the above list was not met by 16 websites who redirected the HTTPS with HTTP/2 request to a plain HTTP website, thereby making the browser fall back to HTTP/1.1 because TBB and other browser vendors only support HTTP/2 when TLS is used. These 16 websites were thus removed from the URL list as a proper comparison of the different HTTP versions could not be made anymore.

In addition to the above, validation of an extra corner case had to be implemented for websites who use Cloudflare *Content Distribution Network* (CDN) as Cloudflare blocks access attempts from Tor exit node IP addresses and forces users to go through a *Completely Automated Public Turing test to tell Computers and Humans Apart* (CAPTCHA) page first [Matthew Prince, 2016][7]. To detect whether a website uses Cloudflare and thus was showing the CAPTCHA page instead of the intended web page the following conditions were checked preceding the general validation; If the first response was a HTTP status code 403 and the response was sent using a server identifying itself as 'cloudflare-nginx'. This was the case for 33 websites in the sample set and also had to be removed form the set.

After pre-processing and validation of the HAR a list of failed attempts was written to a file. The failed attempts were counted per website. Websites were discarded that had failed more than 20 of the 80 iterations. This resulted in another 15 websites being removed from the sample set. Websites that failed less then 20 times were retried in subsequent runs of the crawler until only valid iterations remained. The final list of consisted of 56 URLs, which can be found in Appendix B.

3.4 Feature extraction

Tor multiplexes the data transfer over encrypted channels, so an eavesdropping attacker can only see the direction, size and order of the packets in the stream. These features will be used to train the SVM. The method has been based on previous research done by Wang and Goldberg [2013].

Tor sends all data in padded 512 byte cells (so-called Tor cells), which have been extracted from the pcaps by rounding the record lengths of the packets to the closest multiple of 512 bytes. This number is then divided by 512 to get the number of complete Tor cells in the packet. Each cell is recorded separately, so the Tor cell instances can only contain multiples of 1 in subsequent order. So if a packet contains 3 Tor cells it would become 1,1,1 in the resulting trace. Additionally the size is denoted as positive if it is an outgoing packet, and as negative if it is an incoming packet.

[7] There is still debate whether this actually necessary in the Tor community [Mike Perry, 2016].

Tor uses TLS to encrypt its data, which is designed to encrypt information on the socket layer. It operates independently of TCP/IP, and thus TLS records could be contained in parts of one or more TCP/IP packets. The rounding is necessary because besides Tor cells, there could be a number of bytes of TLS record information as well as empty TLS records used to defend against the BEAST attack [Wang and Goldberg, 2013]. A trace will be made consisting of the aforementioned direction and frequency of the Tor cells. To illustrate[8], consider the following sequence of TCP/IP packets: one outgoing with a length of 544, and one incoming with length of 1088. This will translate to a trace of 1, -1, -1.

When capturing packets from a network interface often packets will be merged if they are larger than the *Maximum Transfer Unit* (MTU). In the case of outgoing packets this is done by the *Generic Segmentation Offload* (GSO) mechanism, and for incoming packets it is done by the *Large Receive Offload* (LRO) mechanism. This could lead to inconsistencies if one would use just the TCP/IP packets as a feature to train a website fingerprinting model on. However, this would have no influence on the traces as used in this research as there would just be more Tor cells within the data part of the merged packets.

3.5 Training the SVM

Website fingerprinting can be viewed as a machine classification problem, like Wang and Goldberg [2013], an SVM has been used in this research as it works well with distance related features [Vapnik and Chervonenkis, 1974]. See Section 5.1 of Wang and Goldberg [2013] of a description on the workings of the SVM in conjunction with the trace data set.

The traces had to be converted to a format that could be used to train the SVM. For this research the implementation[9] of Wang and Goldberg [2013] was used, which in turn is based on the work of Cai et al. [2012]. Their code was able to calculate the *Optimal String Alignment Distance* (OSAD)[10] between the traffic traces. As this process took multiple hundred of *Central Processing Unit* (CPU) hours, it was done on the DAS5 supercomputer [Bal et al., 2016]. Using 10 nodes with dual 8-core CPUs running at 2.4 GHz each, it took for both the HTTP/1.1 and the HTTP/2 data sets about 10 hours in total (5 hours per data set).

The distances then had to be transformed into values suitable for the kernel matrix as described in Section 5.3 of Wang and Goldberg [2013]. The kernel matrix was then split up into the training and test sets which could be used with the SVM. The usage of the code of Wang and Goldberg [2013] allowed for a comparison (see Section 4) between their results and those achieved in this research.

[8] Example taken from Wang and Goldberg [2013].

[9] `https://crysp.uwaterloo.ca/software/webfingerprint/`

[10] Please see Appendix B of Wang and Goldberg [2013] for details on the algorithm.

Chapter 4

Results

As applied by Wang and Goldberg [2013], 10-fold cross validation was done with the SVM on 40 instances of each website of the total 56 websites. This meant that there were 36 training cases and 4 testing cases for each site (224 tests for each fold). For each fold an accuracy was produced, thus 10 per data set. The mean and standard deviation of the 10 accuracy values for the two data sets of the closed-world experiment can be found in Table 4.1. On the columns the HTTP version of the traces that were tested can be found and on the rows the HTTP version used for training the SVM can be found.

The results listed in boldface are of matching HTTP versions for testing and training the SVM. These scores seem to be consistent with earlier findings of Wang and Goldberg [2013] that were done on HTTP/1.1 and resulted in $\overline{x} = 90\%$ with $s = 6\%$. To see whether there is a significant difference in accuracy of the fingerprinting method used in this paper between the different HTTP versions a paired t-test was done, using the results from the optimal scenarios (the bold results in Table 4.1). The result of this paired t-test was a p_{value} of 0.19392 which combined with an α of 0.05 proves that there is no statistically significant difference in accuracy.

Train \ Test	HTTP/1.1	HTTP/2
HTTP/1.1	$\boldsymbol{\overline{x} = 88.036\%\ s = 2.0164\%}$	$\overline{x} = 64.687\%\ s = 6.6631\%$
HTTP/2	$\overline{x} = 54.667\%\ s = 3.5286\%$	$\boldsymbol{\overline{x} = 86.485\%\ s = 3.0871\%}$

Table 4.1: Average accuracy and standard deviation of the 10-fold cross validation.

To see how an SVM trained on a different HTTP version would perform, traces of a different HTTP version were tested on an SVM trained with the opposite HTTP version. The accuracy decreases by approximately 20% and also resulted in a slightly higher standard deviation. For both cases a paired t-test was done, the resulting p_{values} were as follows. The results of HTTP/2 trained SVM tested with HTTP/1.1 (bottom-left) traces in comparison to the optimal HTTP/2 (bottom-right) results the p_{value} was $1.11 \cdot 10^{-15}$. The results of HTTP/1.1 trained SVM tested with HTTP/2 traces (top-right) in comparison to the optimal HTTP/1.1 (top-left) results the p_{value} was $3.58 \cdot 10^{-09}$. With $\alpha = 0.05$ in both cases the difference is statistically significant between the optimally trained SVMs and the SVMs trained with the opposite HTTP version.

Chapter 5

Conclusion

A website fingerprinting attack can be done on a TBB enabled with HTTP/2 in a closed-world scenario. An average accuracy of 86.485% ($s = 3.0871\%$) was achieved on a data set with 56 websites using 40 instances per website. A dedicated attacker eavesdropping on the connection between the client and the guard node will be able to do an accurate prediction of which website a client has visited, as long as the website is in the attacker's trained data.

A comparison between the accuracy of fingerprinting attacks of HTTP/1.1 and HTTP/2 on the TBB has shown that there was no statistically significant difference between the respective average accuracies of 88.036% ($s = 2.0164\%$) and 86.485% ($s = 3.0871\%$). As such, a website fingerprinting attack performs similar for HTTP/1.1 and HTTP/2.

When training with HTTP/1.1 and testing with HTTP/2 an average accuracy of 64.687% ($s = 6.6631\%$) was achieved. When training with HTTP/2 and testing with HTTP/1.1 an average accuracy of 54.667% ($s = 3.5286\%$) was achieved. In both cases the difference was statistically significant when compared with training and testing with the same data set. The accuracy of an attacker's website fingerprinting attack will suffer between 20% and 40% when doing a website fingerprinting attack when the HTTP version differs in his model from the one used by its target.

Chapter 6

Discussion

The results found in this paper were comparable to the work done by Wang and Goldberg [2013]. However, like Wang and Goldberg there were a number of unrealistic aspects of the experiments conducted in this research.

In this paper only a closed-world scenario was examined. This is not realistic since the number of websites on the internet is far more than the 56 used in this research [Juarez et al., 2014]. Wang and Goldberg [2013] and Panchenko et al. [2011] also performed experiments with an open-world scenario. Here websites were randomly distributed into two classes: 'forbidden', and 'other'. This is done to simulate an open-world scenario where an attacker has a list of monitored websites which it tries to recognise in the target's stream. Given a trace of a call to a website, the classifier will try to categorise the testing instance into one of the two classes.

Juarez et al. argued that even the open-world scenario's and experimental setups of Wang and Goldberg [2013] were not modelling a realistic approach of user's browsing habits, differences in versions of the TBB and template websites[1].

Furthermore, Tor is not only used as part of the TBB or for web browsing, but can also be used by other software bundles as a proxy. It can even be used to route all of the traffic of one or more machines through its circuits. A good example of such a setup is Tails[2]. Using Tor not only for web browsing generates other traffic that is mixed together with browser traffic making it more difficult for an attacker to match the traffic to an SVM that is trained to do a website fingerprinting attack based on "clean" training data that has all kinds of "noise" removed resulting in lower accuracy values.

Some websites are more difficult to fingerprint than others due to A/B testing, localisation and rapidly changing content. For example, traces of a news website can be vastly different if they are made a few days apart and page layout has changed. In this case, an attacker would also need to keep his model up to date with the latest version of websites that change their content often. Testing with traces of the newer version of a website will lead to increasingly lower accuracies as the difference between the time the training set and the test were made increases [Juarez et al., 2014].

[1] For example, default themes of commonly used *Content Management System*s (CMSs).

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

In addition, websites which change size often are more difficult to classify correctly [Wang and Goldberg, 2013] [Juarez et al., 2014]. Wang and Goldberg [2013] had found that there were large differences in the size of traces of different languages, and had decided to use localised German versions of web pages to avoid these differences. The authors of this research had tried to achieve the opposite of homogenisation for the sake of realism by using a different circuit for each instance, thereby getting data from multiple exit nodes and thus multiple localised version.

A significant difference was found when training and testing with data from a different set. Future work could look into the effects of training a single model with both HTTP/1.1 and HTTP/2 data.

HTTP/2 has a number of improvements over HTTP/1.1 especially related to multiplexing and prioritised data transfer. The effects of the prioritisation as defence could be examined by randomising the prioritisation of files before each request. The prioritisation could have effect on the order of the Tor cells in the traces and thus might make website fingerprinting more difficult. A similar thing has been tried with HTTP pipelining randomisation[3], however, this was not applied to HTTP/2.

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

Chapter 7

Attributions

Figure 2.1 based on "How Tor Works" images from `https://www.torproject.org/about/overview`.

Devil, Py, Coding, Monitor and Onion icons in Figures 3.1, 2.1 and 2.2 made by Freepik from www.flaticon.com and is licensed by CC 3.0 BY.

Server and Folder icons in Figure 3.1, 2.2 and 2.1 made by Madebyoliver from www.flaticon.com and is licensed by CC 3.0 BY.

Acronyms

AJAX *Asynchronous JavaScript and XML*. 9

CAPTCHA *Completely Automated Public Turing test to tell Computers and Humans Apart*. 11

CDN *Content Distribution Network*. 11

CMS *Content Management System*. 15

CPU *Central Processing Unit*. 12

CSV *comma-separated values*. 9

GSO *Generic Segmentation Offload*. 12

HAR *Hypertext Transfer Protocol Archive*. 8, 10, 11

HOL *Head-of-line blocking*. 5

HTTP *Hypertext Transfer Protocol*. 1, 2, 4, 6–8, 10, 11, 13, 14, 16, 18

HTTP/1.1 *Hypertext Transfer Protocol 1.1*. 1, 2, 4–6, 8, 10–14, 16

HTTP/2 *Hypertext Transfer Protocol 2*. 1, 2, 4–14, 16

HTTPS *HTTP over TLS*. 7, 11

IP *Internet Protocol*. 7, 9–11

IP address *Internet Protocol address*. 4

JSON *JavaScript Object Notation*. 8

LRO *Large Receive Offload*. 12

MTU *Maximum Transfer Unit*. 12

ORport *Onion Routing port*. 9

OSAD *Optimal String Alignment Distance*. 12

pcap *packet capture*. 8, 10, 11

SVM *Support Vector Machine*. 1, 3, 6, 7, 10–13, 15

TBB *Tor Browser Bundle*. 1–4, 6–9, 11, 14, 15

TCP/IP *Transmission Control Protocol / Internet Protocol*. 1, 4–6, 12

TLS *Transport Layer Security*. 4, 5, 7, 11, 12, 18

Tor *The Second-Generation Onion Router*. 1–6, 8–12, 15, 16

URL *Uniform Resource Locator*. 1, 2, 7, 9–11

Bibliography

Henri Bal, Dick Epema, Cees de Laat, Rob van Nieuwpoort, John Romein, Frank Seinstra, Cees Snoek, and Harry Wijshoff. A medium-scale distributed system for computer science research: Infrastructure for the long term. *Computer*, 49(5):54–63, 2016.

M. Belshe, R. Peon, and M. Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540 (Proposed Standard), May 2015.

Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 605–616. ACM, 2012.

Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

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

Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 263–274. ACM, 2014.

Marc Liberatore and Brian Neil Levine. Inferring the source of encrypted http connections. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 255–263. ACM, 2006.

Cloudflare Matthew Prince. The trouble with tor. `https://blog.cloudflare.com/the-trouble-with-tor/`, March 2016. (Accessed on 30/01/2017).

Torproject Mike Perry. The trouble with cloudflare. `https://blog.torproject.org/blog/trouble-cloudflare`, March 2016. (Accessed on 30/01/2017).

Ricardo Morla. An initial study of the effect of pipelining in hiding http/2.0 response sizes. *arXiv preprint arXiv:1607.06709*, 2016.

Steve Ouders. Roundup on parallel connections. `https://www.stevesouders.com/blog/2008/03/20/roundup-on-parallel-connections/`, 2008. (Accessed on 08/02/2017).

Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pages 103–114. ACM, 2011.

Vladimir N Vapnik and Alexey J Chervonenkis. Theory of pattern recognition. 1974.

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*, pages 201–212. ACM, 2013.

IETF HTTP WG. Http/2 frequently asked questions. `https://http2.github.io/faq/#does-http2-require-encryption`, 2017. (Accessed on 08/02/2017).

Appendix A

TBB configuration

The configuration of the TBB can be found in Table A.1. All settings that are not listed here are set to their default values. tbselenium also adjusts the config to allow it to connect to the Firefox webdriver, these changes can be found in the code on its repository: `https://github.com/webfp/tor-browser-selenium`.

Table A.1: Configuration of the TBB, with the additional lines only used in the HTTP/2 experiments in bold.

Name	**Value**
network.http.spdy.enabled	true
network.http.spdy.enabled.http2	true
network.http.spdy.enabled.http2draft	true
network.http.spdy.enabled.v3-1	true
devtools.netmonitor.har.includeResponseBodies	false
extensions.netmonitor.har.enableAutomation	true
extensions.netmonitor.har.contentAPIToken	test
extensions.netmonitor.har.autoConnect	true
browser.cache.disk.enable	false
browser.cache.memory.enable	false

Appendix B

URLs

Table B.1: List of URLs used in experiments.

alibaba.com	merdeka.com
aliexpress.com	messenger.com
asana.com	myanimelist.net
blogfa.com	nytimes.com
blogger.com	reddit.com
blogspot.com	researchgate.net
blogspot.in	rutracker.org
blogspot.jp	shink.in
blogspot.mx	shopify.com
blog.wordpress.com	slack.com
coursera.org	softonic.com
detik.com	spotify.com
diply.com	stackexchange.com
discordapp.com	stackoverflow.com
doubleclick.net	t.co
dropbox.com	torrentz2.eu
facebook.com	trello.com
files.wordpress.com	twitter.com
fmovies.se	ultimate-guitar.com
gfycat.com	vk.me
goo.gl	wikimedia.org
google.com	wikipedia.org
gyazo.com	wiktionary.org
instagram.com	wordpress.com
kaskus.co.id	wordpress.org
kooora.com	wp.pl
linkshrink.net	xhamster.com
medium.com	youtube.com