Website Fingerprinting on Tor With Machine Learning

Reid M. Bixler
Computer Science Dept.
Virginia Tech
Blacksburg, VA 24060

Abstract

In this paper, I analyze the effects of a range of machine learning algorithms for the purpose of website fingerprinting attacks on the Tor network. The Tor browser is meant to mask the identity of the user, but there is the possibility of information being leaked despite being encrypted. The intention of this paper is to look at both the open- and closed-world scenarios of attack with the hope of building some machine learning models that can perform better fingerprinting than previous models as well as propose potential defenses against such attacks.

1 INTRODUCTION

The Tor network is an alternative means to connect to the internet that allows the user to be able to mask their identity through a series of security techniques called onion routing. This onion routing is an efficient and secure way of preventing on-the-line snooping and networking attacks when connecting to internet, utilizing public key cryptography and multi-tunneled traffic to prevent such attempts from succeeding (Dingledine et al., 2004).

While the Tor browser is quite secure in comparison to normal browser traffic, it is not impervious to all methods of attack. One such attack, website traffic fingerprinting, lets attackers figure out what sites are being accessed by a Tor user, to the extent that it removes a major portion of the anonymity provided by the Tor browser (Panchenko et al., 2011).

Previous attacks have looked at timing and counting the connections, but the introduction of OpenSSL's TLS records and Tor's use of TCP pushback to do rate limiting means that tracing by IP packets produces very poor results (Dyer et al., 2012). As such, I have implemented some website fingerprinting attacks that do not rely soley on these features. Website fingerprinting on the Tor browser, while not impossible, can be difficult to carry out, but it can be even more difficult to come up with results that arent limited by OpenSSLs TLS records and Tors use of TCP pushback for rate limiting (Dingledine et al., 2006).

One of the key aspects that must be taken into consideration when considering website fingerprinting attacks is which of the 2 different types of scenarios the attack is taking place in: closed-world and open-world (Panchenko et al., 2011). Closed-world means that the researcher is trying to identify websites out of a known set of possible websites that the user could be accessing, for times when Tor is being used infrequently for only a few sites. Open-world, a far more difficult and realistic scenario, is trying to identify a couple of pages out of a very large set of unknown websites.

While I would definitely like to have my method of fingerprinting be applicable to a real-world scenario, there is a very real possibility that the real-world scenario is too difficult of a scenario for our modern classifiers because of the exponential increase in difficulty when introducing all possible websites a user could access with the Tor browser.

In my experiments, I test both the closed- and open-world scenarios in attempt to figure out the best possible option for both scenarios, but it is to be expected that the closed-world scenario will likely result in much better attacks due to the limited scope of websites that could be accessed (Cai et al., 2012).

While attack is one large part of this research paper, I have also taken a look at possible defenses against these attacks (and potentially other attacks). Security is paramount for the usability of the Tor browser, and an introduction of a potentially harmful fingerprinting attack without defense would leave a number of Tors users open to such attacks. As such, I go about implementing some of the suggestions

that the authors of Tor think would be good defenses for my attack(s): changing Tors cell size, padding techniques, or traffic delays (Cai et al., 2014).

In the best case scenario, a defense will cover all for possible attacks, but there is always the possibility that a defense is realistically only useful for some attack scenarios that focus on certain aspects of the Tor connection. As such, there is a case where Tor or the users of Tor must implement multiple defenses to protect against all possible attacks(Abbott et al., 2007). In the worst case scenario, though, there may exist a website fingerprinting attack that focuses on an unfixable portion of the Tor browser, essentially something that is inherent and required for the Tor browser to work such that a modification to it is not possible.

I have built a number of different classifiers based off of downloading the sites from Alexas top 500 to generate signatures for each website in an attempt to fingerprint each of these websites when accessed over the Tor browser. There are a number of features that can be used outside of the timing and size of the TCP packets, including the actual TLS records themselves as well as other connection information sent during every Tor connection.

This approach has been more similar to a scatter-shot technique, since I did not know which features would prove to be more useful than others. As such, my intentions were not to prove whether one specific attack worked, but rather testing the validity of a gambit of attack vectors and then analyzing if any of them were able to produce results good enough to be able to consistently fingerprint a users traffic over the Tor browser.

1.1 RELATED WORK

There have been a wide variety of attempts at website fingerprinting in the past, some of which have had a similar approach as to my own using machine learning models. An earlier example of this was using Hidden Markov Models on packet traces in order to defeat the previously secure application-level defense of HTTPOS and randomized pipelining over Tor (Cai et al., 2012).

A similar approach looked at using Support Vector Machines (SVM) on the more fundamental Tor cells as a unit of data rather than TCP/IP packets (Wang and Goldberg, 2013). This approach however studied a broken implementation of the Tor browser despite the authors of Tor offering to assist in the work.

Another paper by some of the same authors, the researchers had looked at using the k-Nearest Neighbors (KNN) algorithm on packet sequences in order to fingerprint a range of websites (Wang et al., 2014).

The problem with most if not all of these previous works were that they relied on the timing and counting of the TCP connections, rather than other possible features contained within every connection through the Tor browser. These approaches have used fairly simplistic models: Hidden Markov Models, Support Vector Machines, and k-Neural Networks, which is reasonable considering the problem domain. However, there have been a number of advancements within the machine learning realm that could mean the creation of alternative classifiers that would be able to better classify website signatures for the website fingerprinting attack.

2 BACKGROUND

When analyzing the potential of website fingerprinting attacks with machine learning algorithms, one must first look at what it means to use such algorithms for classification. As such, classification accuracy relies on 4 key factors: the size of the hypothesis space, the accuracy of the feature extraction, the size of the instance space, and the number of training examples provided (Michalski et al., 2013). Specifically, it would be expected that as the number and/or complexity of classification categories increases while reliable feature information does not, the classifier eventually runs out of descriptive feature information, and either true positive accuracy goes down or the false positive rate goes up (Perry, 2013).

Specifically for the case of website fingerprinting, we must consider the problem presented by false positives within machine learning models. False positives generally will ruin the effectiveness of classification unless the values are exceedingly small (much smaller than 10^{-6}). In the case of website fingerprinting, false positives can be extremely damaging to these types of attacks more so than other domains. If any page has similar aspects to that of another page enough to trigger a false positive, then the whole set of pages are generally going to be misclassified (Perry, 2013).

Each page of a website actually also has at least 8 different common traffic patterns consisting of the combination of the following common browser configurations: cached vs non-cached, javascript enabled vs disabled, and adblocked vs non-adblocked (Perry, 2013). However, it must be noted that these don't necessarily describe the webpage as a whole, but rather could be defining individual elements within the page. Therefore, each element could be an individual feature that might be used to discern from other pages. That being said, simple defenses within the Tor browser tend to mask these traffic patterns leaving the model with little to learn on.

When considering false positives as a key factor of the

website fingerprinting attacks, It is likely that in practice, relatively simple defenses will still substantially increase the false positive rate of this attack when it is performed against large numbers of web pages (Perry, 2013).

3 APPROACH

For this paper, I have defined 4 key evaluation questions that I intended to answer from the outset of my research:

1. Is it possible to use a machine learning model to fingerprint websites over the Tor browser?
2. If so, what type of machine learning models prove more beneficial than others?
3. Given the attack is possible, what types of defenses could combat the given attack(s)?
4. Assuming all of the above, will some defenses not work for some machine learning models but work for others?

With these questions in mind, the general approach to my machine learning models was to first see what I needed to do different than what other previous works had done. Firstly, I wanted to focus on the TLS records themselves (headers, # of 512 byte cells, combined features, etc.). Similarly, I wanted to try to extract as many features as possible, under the assumption that more features should result in less false positives. Finally, I wanted to also see if there were ways to focus on fingerprinting the individual websites that tended to stick out in my experiments and see what specifically it was that made them easy to identify in the first place.

For this work, I used a couple of tools for network analysis and general usage of Tor: the Tor browser, Wireshark, and tcpdump. My goal was to analyze Alexa's Top 500 United States website traffic from my computer as my 'dataset' and then to train my machine learning models based off of the encrypted traffic sent over the Tor network.

My general approach for what I wanted to do was to first recreate previous experiments/works by implementing some simplistic machine learning models trained on the basic features inherent in the packets to see what type information that I could glean with the most basic setup. Secondly, I wanted to generate a number of new features such as the distribution of packets, rate at which the packets were being sent, and what the count of the first or last certain amount of packets were. Once these features were extracted, the next step was to generate some alternative classifiers that were perhaps a bit more complicated than previous works, utilizing the newly generated features, with the hopes of discerning more information than the basic setup could obtain.

4 EXPERIMENTAL SETUP

For my first set of experiments, I trained 5 different machine learning models: K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Hidden Markov Models (HMM), Logistic Regression, and Stochastic Gradient Descent (SGD). All of these were using some machine learning libraries within Python that implemented all of these algorithms/methods outright. The features that were focused on in this first set of experiments were: total number of bytes, total number of packets sent, and ratio of incoming vs outgoing packets. The dataset used was based off of Alexa's Top 500 United States websites in both a closed-world and open-world setting. In this case, the closed-world experiment meant training on all 500 websites through accessing the individual webpages as signatures and then using those signatures to train on. For testing the closed-world scenario, I tested on a subset of 50 of these websites a few times. For the open-world experiment, I similarly trained on all 500 websites and then tested on 100 random websites with at least 25 of those websites being from the original training set.

For the second set of experiments, I trained 3 new machine learning models: Principal Component Analysis (PCA), Q-Learning, and Kernel Approximation. Similarly to before, I used some machine learning libraries within Python that had already implemented these algorithms for general usage. In this case, however, the features were split into 2 sets. Set 1 had the same features as that of the first set of experiments (# bytes, # backets, ratio of packets). Set 2 had the same features as Set 1 but also included: the distribution of the packets, the rate at which the packets were sent, the first and last 10 packets, the first and last 5 packets, the mean time taken, and the standard deviation of time taken. Both the closed- and open-world approaches were the same in terms of training and testing approaches.

5 RESULTS

5.1 Experiment 1

In the case of Experiment 1, it's possible to see that in both the closed-world and the open-world scenarios that I did not ended up achieving anywhere near a tolerable false-positive rate (0.02%) considering the size of the datasets. Even in the cases when it was relatively low, the overall detection rate (true-positives) was not generally good enough in comparison to previous works which were well above 50% detection within the closed-world

scenario at least.

Closed-World Scenario

	KNN	SVM	HMM	LR	SGD
TruePos	32%	38%	34%	30%	34%
FalsePos	0.13%	0.21%	0.09%	0.14%	0.18%

It's notable that, as expected, the open-world scenario does tend to result in slightly worse true positives likely due to the fact that the size of the domain increases and the likelihood of similar looking webpages increases greatly, especially considering those that have not been seen by the models before. One should also note that the disparity between algorithms within a few percentage points is not enough to warrant stating that one is better than another, as this is most likely simply contributed to either the training process or the randomization when picking the testing set.

Open-World Scenario

	KNN	SVM	HMM	LR	SGD
TruePos	26%	28%	32%	28%	28%
FalsePos	0.17%	0.19%	0.13%	0.11%	0.22%

Based off of these results, it became apparent that I needed to go about approaching this from a different angle, by going onto my second experiment. The hopes of this were to utilize some potentially more powerful machine learning algorithms as well as use more features to be able to learn more specific details that may not have been picked up in the most basic setup.

5.2 Experiment 2

From running this new set of experiments, it became apparent that introducing more complicated classification algorithms would not contribute any more accuracy or decrease in false positive rate, and in some instances could actually produce worse results (in the instance of kernel approximation).

Closed-World Scenario - Set 1

	PCA	QL	KA
TruePos	32%	36%	28%
FalsePos	0.14%	0.13%	0.25%

Notably, and as expected, introducing more features into the machine learning models tended to result in slightly better classification accuracy, but there was generally not enough change within the false positive rate to be considered as a result of introducing more features. There may be some possibility that the false positive rate was actually affected, but the experiments that I ran were not in-depth enough to be able to calculate whether or not there was such a change.

Closed-World Scenario - Set 2

	PCA	QL	KA
TruePos	46%	42%	38%
FalsePos	0.12%	0.08 %	0.22 %

I have chosen not to show the results of the Open-World Scenario for the second experiment, mostly because it sees almost the same exact results as the Closed-World Scenario. This being that the increase of features increases accuracy and that the more complicated models themselves do not attribute towards any benefits.

6 DISCUSSION

6.1 DEFENSES

The Tor browser itself actually includes built-in security which can be modified by the user. These Tor Browser Security Settings are provided in 3 different levels, with 1 being the default and lowest level of security (which I ran while running my experiments). Level 1 security has all browser features enabled. Level 2 security implements: HTML5 video and audio media become click-to-play via NoScript, on sites where JavaScript is enabled, performance optimizations are disabled, scripts on some sites may run slower, some mechanisms of displaying math equations are disabled, some font rendering features are disabled, and JavaScript is disabled by default on all non-HTTPS sites. Level 3 security implements: JavaScript is disabled by default on all sites, some types of images are disabled, and some fonts and icons may display incorrectly.

However, it is also possible to defend Tor's browsing with exterior defenses that are not implicitly provided with the service. Things such as changing Tor's cell size from the original 512 bytes to 1024, introducing padding techniques, traffic delays, loading a background page, and randomized pipelining. All of these defenses are meant to be potentially used to prevent different parts of the browsing to be masked or difficult to extract features from.

When implementing these defenses and re-running the previous experiments, nearly all classification accuracies decrease with each increase in security level in the Tor browser. Similarly, any defense turns an already bad detection rate to even worse, with an average 14% decrease in accuracy across all models/features. The most effective defense, in the case of my 'attacks', seemed to be padding for fixed or random packet sizes, resulting in a nearly 20% decrease on average for all models. This is likely due to the fact that my models' features tended to rely on the packets for extracting information.

One thing of note is that these defenses can also be implemented *on top of one another* as a layered defense. These layered defenses in general resulted in even worse classification than before, or even not being able to classify at all in some cases. Essentially, by layering the defenses it was possible to make any previous result (which were already shown to be lackluster) become basically useless with enough defense.

6.2 ANALYSIS

The big question based off of these results therefore is: Why did these classifications fail? After analyzing the approaches, the classifications, and the general experiments there appears to be a few key things that caused these problems in my experiments. First and foremost, the size of the hypothesis space tended to be far too difficult considering that there tended to be too many similar websites. These similar websites in turn would result in a false-positive rate well above an acceptable threshold for this type of attack. Secondly, the feature space was very limited, seeing as one can only interpret some encrypted web traffic so many different ways. For what few discernable features there may have been in websites via caching/JavaScript, any feature that may have provided an increase in accuracy initially, was easily defended against.

Notably, while my initial thoughts that more complicated machine learning models would result in better results, I tended to find nothing that pointed towards that in the end. After looking more in depth into these algorithms, I began to realize that these algorithms tended to benefit from **very** large instance spaces or number of training examples. In my case, it simply was not feasible to train and test at the scale of some of these previous works which were in the order of millions.

Another key thing that I noticed that even being able to fingerprint a website that are drastically different from others (i.e. ones that load videos/large data files) does not necessarily mean that it is possible to identify specifically *what* is being accessed on that website. While it may be useful to identify websites such as Youtube or Netflix due to the content they provide on their landing pages, generally it is not useful since there is no discernable information on top of that.

6.3 CONCLUSION

Having seen these results of my research there were a number of takeaways that may benefit were I to do similar research again. Most importantly, it would be that feature extraction and feature selection are absolutely key for making sure your models work correctly. While one may initially think that a fancier model would result in better results, it's rather that finding more discernable features in your dataset will almost always tend to result in better accuracy in the long run. Secondly, it would be better to focus on the minimization of the false positive rate rather than increasing the detection rate. This is due to the fact that any false positive rate above a reasonable threshold pretty much means that there's a high probability that you end up 'accusing' a user of accessing a website that they did not access.

Another takeaway was that training and testing with the defenses **already** in place will contribute towards easier model iteration in the long run (assuming that the features are good enough to begin with). While this may initially lead to low detection rates, if you instead focus on trying to train models that are good at finding the 'simple' things in the data then by the time the defenses are implemented that remove those 'simple' things then the models themselves have nothing to hold onto. Instead, if the models were to be trained intially with the defenses in place, it may be more difficult initially to move forward but it's more likely to result in useful results. Finally, having a larger hypothesis space (e.g. training/testing on more websites) would generally be better in order to be able to discern more differences between the webpages.

References

Abbott, T. G., Lai, K. J., Lieberman, M. R., and Price, E. C. (2007). Browser-based attacks on tor. In *International Workshop on Privacy Enhancing Technologies*, pages 184–199. Springer.

Cai, X., Nithyanand, R., Wang, T., Johnson, R., and Goldberg, I. (2014). A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 227–238. ACM.

Cai, X., Zhang, X. C., Joshi, B., and Johnson, R. (2012). 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.

Dingledine, R., Mathewson, N., and Nerad, S. (2006). Tor development roadmap: Wishlist for nov 2006–dec 2007.

Dingledine, R., Mathewson, N., and Syverson, P. (2004). Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC.

Dyer, K. P., Coull, S. E., Ristenpart, T., and Shrimpton, T. (2012). Peek-a-boo, i still see you: Why efficient

traffic analysis countermeasures fail. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 332–346. IEEE.

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (2013). *Machine learning: An artificial intelligence approach*. Springer Science & Business Media.

Panchenko, A., Niessen, L., Zinnen, A., and Engel, T. (2011). 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.

Perry, M. (2013). A critique of website traffic fingerprinting attacks.

Wang, T., Cai, X., Nithyanand, R., Johnson, R., and Goldberg, I. (2014). Effective attacks and provable defenses for website fingerprinting. In *USENIX Security Symposium*, pages 143–157.

Wang, T. and Goldberg, I. (2013). Improved website fingerprinting on tor. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 201–212. ACM.