Large-scale Website Fingerprinting

Eric Jollès, Spring 2020, SPRING Laboratory, EPFL

Abstract

Being anonymous when we browse a website is essential to preserve freedom of speech and information. For that purpose one can use anonymous network, such as Tor, which encrypts data and preserves user anonymity. However, the metadata of exchanged packets is not hidden. In website fingerprinting attacks, a passive attacker uses this metadata to predict, with machine learning techniques, which website was accessed by one user. These attacks, if deployed on a large scale, may allow governments or Internet Service Providers (ISPs) to monitor communications, and therefore threaten the privacy of users. In this paper, we present an implementation of a large-scale website fingerprinting attack.

1 Introduction

In standard web communications (HTTP or HTTPS), an eavesdropper can observe the source and destination IP-address of each packet. A server having access to these addresses, can easily deanonymize a user. To counter this, users can use Tor [9], which is a network providing anonymity.

However, information such as the timing, size, and direction of transmitted packets is not hidden at the IP layer. Combined with the fact that each website has its own network traffic, one can mount a Website Fingerprinting (WF) attack to estimate which website is currently being browsed. WF attacks can be used for many purposes such as identifying every web page a user visits, blocking access to certain websites or clustering users visiting targeted websites.

Previous work demonstrates the effectiveness of WF attacks on Tor [10, 3]. These attacks use datasets collected once with the attacker's computer. However it has been proven that these attacks have an accuracy deeply affected by the freshness of the data [5]. In order to make predictions in real-world scenarios, we need to find a way to continuously update our classifier with recent data.

In this project, we develop a Large-scale Website Fingerprinting attack. We propose a new way to automatically and continually collect a large amount of packet sequences. We also provide modifications to existing attacks so that they can handle as many packet sequences as possible while making as few prediction errors as possible.

2 Preliminaries

2.1 Tor network

With Tor, when a client wants to contact a server, a circuit of 3 intermediate servers (nodes) is created. Packets are sent to the server through these nodes so that no one (except the client) can know the identity of the client and server at the same time. One node only knows its previous and next node of the circuit.

2.2 WF attacks

In state-of-the-art WF attacks, an attacker collects communication packets as he visits a set of known websites. A feature set can be obtained by extracting precise metadata from the packet sequences. This feature set can then be used to train a machine learning (ML) classifier. The attacker can afterwards intercept communications between the client and the first Tor node – called guard node – to make predictions.

2.3 Open-world vs. Closed world

WF attacks are not usually trained with all existing websites, as their number is constantly increasing, it also could not be scaled up. Attacks are trained on a set of known websites, chosen by the attacker, which we call monitored websites.

Then when an attacker wants to make predictions, he can be in 2 different scenarios:

- a closed-world scenario, in which we assume each packet sequence belongs to the set of monitored websites. It is often used to evaluate WF security defence.
- an open-world scenario, in which a packet sequence can belong to the set of monitored websites or unmonitored websites (websites unknown by the attacker). This is closer to a real-world scenario and is often used to evaluate the scalability of an attack.

3 Related work

3.1 Existing WF attacks

Through this project we compare several WF attacks which we will now itemize. A WF attack is defined by a feature set and a ML classifier.

Wang et al. attack

In this attack there are more than 3000 extracted features, whether general features (transmission size or number of incoming/outgoing packets) or more specific features such as unique packet lengths, concentration of outgoing packet

etc. The corresponding machine learning model used is the well known k-nearest neighbor (k-NN) classifier with weight adjustments [10].

Hayes et al. attack (k-fingerprinting attack)

The feature set also contains information about the timing and size of packet sequences. However these features are processed through two different machine learning classifiers: first a random forests (RF) needs to extract a fingerprint for each traffic instance, which is then fed to a k-NN [3].

Triplet Fingerprinting (TF) attack

The triplet fingerprinting attack is using neural networks (NN) in order to extract features. This neural network tries to minimize the distance between 2 samples referencing to the same website while maximizing the distance for examples referencing different websites. The distance metric used is the cosine distance[1].

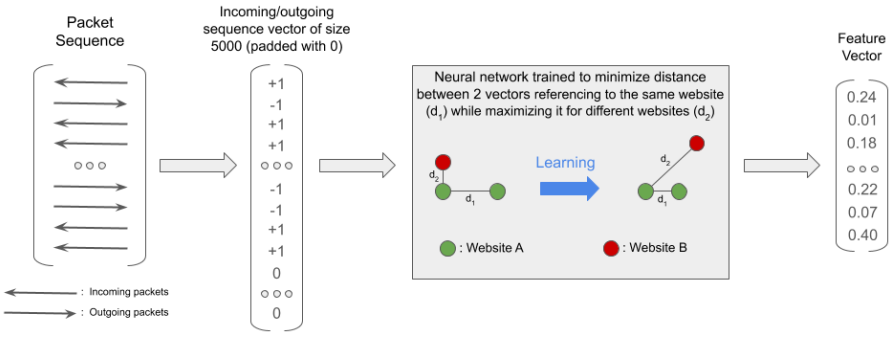


Figure 1: Features extraction of the triplet fingerprinting attack.

As shown in Figure 1, the TF attack uses sequences of incoming and outgoing packets of the first 5000 packets exchanged. These sequences are then fed to the neural network (NN) to obtain the feature set. A k-NN classifier uses these features to make predictions [8].

The approach in the TF paper has shown better results, in terms of time and accuracy, by first averaging out the NN output related to one website and then use a 1-NN classifier with these mean embedded vectors (TF-MEV).

The size of the feature set for each attack is summarized in Table 1.

[1]The cosine distance between a vector $\vec{X}$ and a vector $\vec{Y}$ is defined as $1.0 - \frac{\vec{X}.\vec{Y}}{||\vec{X}||||\vec{Y}||}$

Attack name	Number of extracted features
Wang	3736
Hayes	175
Triplet	64

Table 1: Number of extracted features per WF attack.

4 Large-scale data collection

In previous work, researchers evaluate attacks with synthetic data, i.e. collected on their laptops, once and on a limited number of websites. However, in a large-scale WF attack, we want to have a lot of recent packet sequences from many websites.

We thus propose to automatically and continuously extract data from the last node (exit node) of the Tor circuit (see Figure 2) in order to update the training dataset. Since the Tor exit node knows the website IP address in packet exchanges, the attacker gets fresh data with the associated website.

For the attack (prediction) phase we will take packets from the first node (guard node) of the Tor circuit. From the guard node, the attacker knows the user IP address. So he can use the attack and the exchanged packet sequences to predict which websites are visited by any users.

A large number of packet sequences are permanently extracted from the guard and exit node. So if the WF attack used is not fast enough, some packet sequences will be dropped. We thus need to find a way to reduce the time required to update our training dataset and to make website predictions while maintaining a good classification accuracy. That will be done in section 5.

We assume that the attacker is passive, i.e. he can only read packet sequences and cannot modify or remove them. He also does not know the server IP address and cannot read the content of the exchanged packets

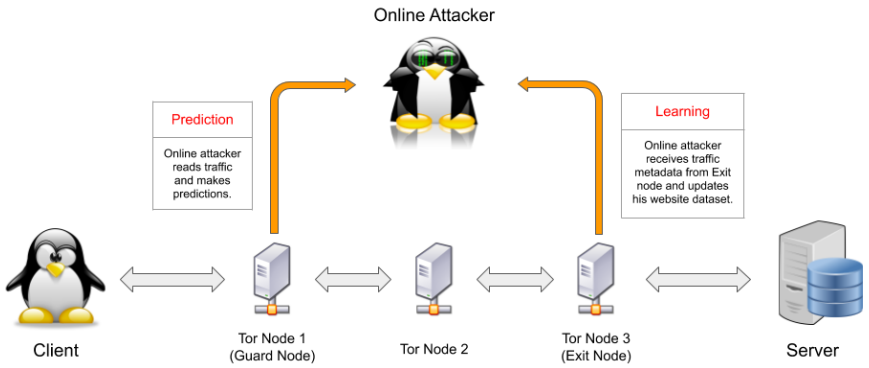


Figure 2: Our large-scale WF attack structure.

5 Attack improvements

The main bottleneck of current attacks lies in the prediction time. The time required to make a prediction using the k-NN algorithm grows linearly with the size of the training set and with the dimension of each datapoint. If we keep every collected data points, as in existing attacks, we could have important execution times. Thus we need to reduce the number of data points while keeping enough packet sequences to keep our system reasonably efficient.

In our online implementation, 3 metrics are important:

- the execution time required to update the training data.
- the execution time required to classify a packet sequence.
- the cumulative error, which shows the evolution of prediction errors. Once a prediction is made with a data point, this data point is given to the classifier as a training point.

In this section we will itemize the different implementations we tried and their corresponding results.

5.1 Queue

We first consider a queue implementation, in which at most q data point are stored for each website (label), where q corresponds to the queue size. Since queues use the first-in-first-out policy, we only keep the most recent data points in memory. As shown in Figure 3, the queue will contain the most recent features extracted from packet sequences. All data points of these queues become the training point of the classifier when a prediction is made.

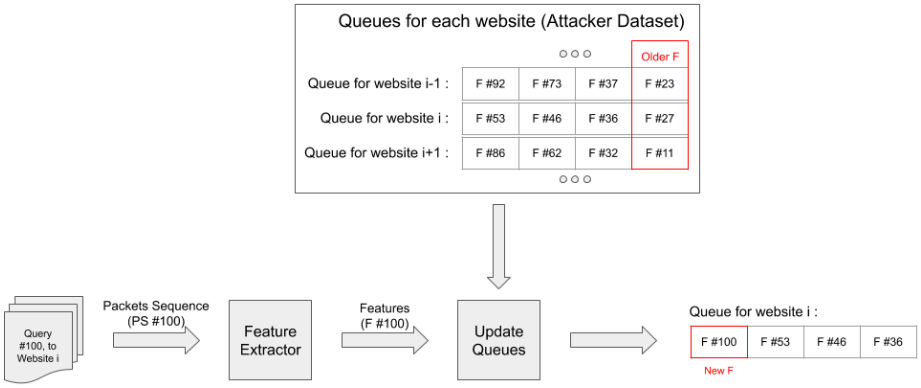


Figure 3: Queue update mechanism with queue size 4. Here F stands for Features and PS for Packet Sequence.

5.2 Incremental learning

In the queue implementation, each time we make a prediction, we need to fit all data in our ML classifier (in our cases k-NN). In some libraries, as the one we used : scikit-learn [6], this step is very time consuming.

We thus tried to use an incremental approach to train the ML classifier, which does not need to fit all data for every prediction. These models have the advantage to be quickly updatable and produce also fast classification. For this purpose, we used Creme [2], which is a python library for online machine learning.

5.3 Average Point

For the Average point implementation, instead of keeping a queue of q elements per website, we decided to keep one data point, which will be the mean of all data points linked to the website.

6 Results

In this section, we use a closed-world scenario, with 2 datasets often used for evaluating WF attack:

- Wang's dataset [10] containing 100 monitored websites selected from a list of blocked web pages in different countries. Each of these websites has 100 different packet sequences. The unmonitored websites are selected from Alexa's top 10,000 websites[1].
- DF dataset [7] which is way bigger. It contains 1,250 packet sequences for each of the top Alexa 100 websites [1]. Unmonitored websites are selected from Alexa's top 50,000 websites.

The machine on which we run the experiments has an Intel Xeon Gold 6240 CPU @ 2.60GHz, a Tesla V100-SXM2-32GB and 400 GB RAM.

6.1 Queue

We trained the TF neural network once before the experiments. In Wang's attack, weights used for k-NN classifier are computed once before making prediction. However, if we update the dataset, they need to be recomputed, which takes too much time (about 10 minutes for each prediction). We have the same problem with the Random forest (RF) used in Hayes' attack. So we also decided to compute them once before doing the experiments.

With the queue implementation, the model update time (time to update the dataset) includes feature extraction time as well as queue update time. The most time consuming operation is the feature extraction.

We can see in Figure 4 that with more than 3000 features to extract, Wang's attack is by far the slowest. Hayes and TF-MEV attack, on the other hand, have

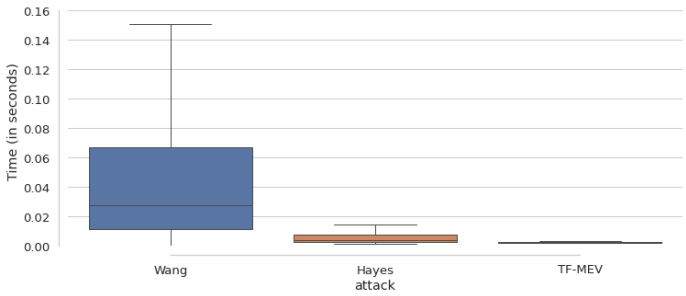


Figure 4: Time for one feature extraction.

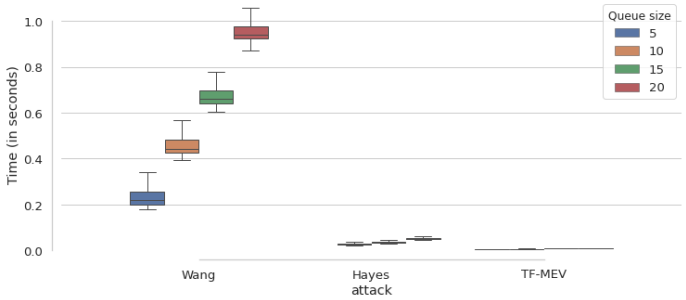


Figure 5: Time for one prediction.

smaller feature sets (see table 1), and thus have more reasonable times, on the order of a hundredth of a second.

We notice a similar trend in Figure 5 for the prediction time which includes the feature extraction time as well as the execution time of the classifier. TF-MEV is the fastest since it feeds the k-NN with only one data point per website.

Fewer data points means less information available, so each classifier gives poorer predictions than with all data points. Furthermore, with a fixed RF, Hayes' attacks gives really poor results. Only TF seems to give good results for small queue size as we can see in Figure 6.

TF-MEV seems to be the most suitable WF attack when a small number of data points is kept, so we decided to continue our experiments with this attack.

6.2 Incremental learning

The results are a little mixed.

Indeed Creme uses a moving windows for k-NN, which corresponds to a global queue, whereas in our previous implementation, we used one queue per website. We thus have to adapt the TF-MEV attack to Creme's k-NN, which impacts the prediction time (see Figure 7). Creme's benefits are thus lost.

Creme's implementation also gives more errors (see Figure 6) since we can-

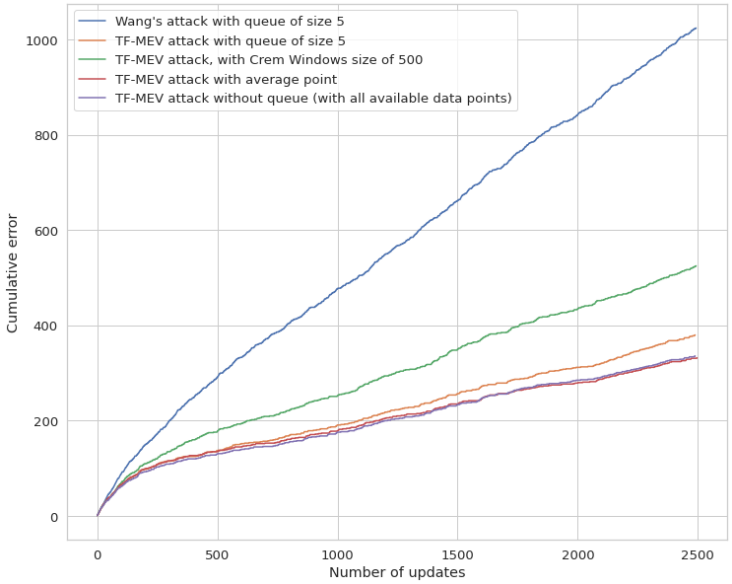


Figure 6: Cumulative error for different WF attacks. Hayes' attack was withdrawn for readability reasons.

not, with Creme, choose the appropriate distance metric (cosine distance for TF-MEV). With the Creme moving window, we can also have missing monitored websites in the dataset depending on the size of the window, which was not the case in our queue implementation.

6.3 Average point

As the mean computation is done during the update, the prediction is faster than the queue implementation (see Figure 7). Furthermore when averaging all packet sequences, we keep information from each packet sequence. As shown in Figure 6, the final incremental error is thus close to the TF-MEV attack which uses all available packet sequences.

6.4 Open-world results

In a real world scenario, during prediction, it will often happen that the website linked to the packet sequences does not correspond to one of the target classes of the classifier. In order not to always obtain mispredictions in this case, we need to create a classifier to predict whether a website is monitored or not.

In the TF attack under the open-world scenario, the unmonitored websites were included as an additional website during training. This attack gives good results with existing datasets, however it is not a good enough solution. Indeed,

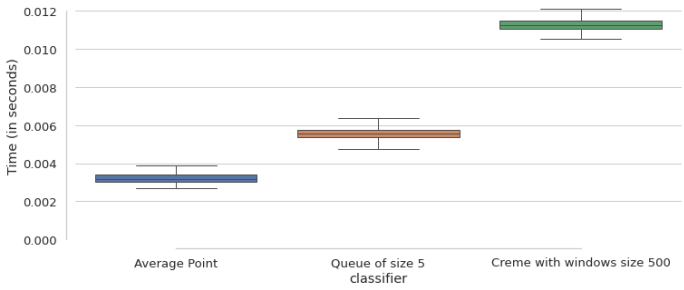
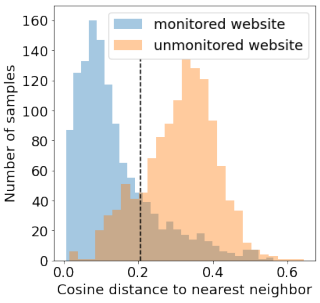


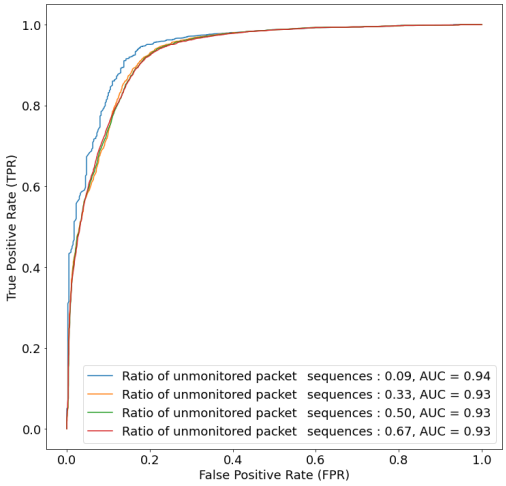
Figure 7: Time of 1 prediction with TF-MEV attack.

if we have a website that appears very often in the unmonitored set, the attack will learn how to classify this website and not unmonitored websites.

We thus wanted to create our own classifier. As we can see in Figure 8a, the distance to nearest neighbor is smaller with monitored websites than with unmonitored websites. We thus need to find a threshold to predict whether a point belongs to the set of monitored website. We can see in Figure 8b that our classifier has good performances, and that the ratio of unmonitored data does not change these performances. The threshold must thus be finely tuned in function of the unmonitored website distribution (in our test with Wang's dataset, the threshold is set to 0.2).



(a) Distance distribution to nearest neighbor for TF-MEV attack with 2500 data point of Wang's Dataset. The dashed line corresponds to the chosen threshold (0.2) for the monitored/unmonitored website classifier.



(b) ROC for the monitored/unmonitored classifier of the TF-MEV with Wang's dataset.

Figure 8: Results of TF-MEV with Average Point in an open-world scenario

7 Large-scale WF attack implementation

For our final implementation we finally chose to use the Average Point since it provides the best results. We decided to create a server, on which we could execute the attack. A Tor exit node sends packet sequences to the server in order to update the server dataset. An attacker can then ask the server to predict one website with some packet sequences. This server communicates through TCP sockets. We have results similar to the Average point implementation for local tests, However these results should vary with the addition of transmission times. Note that the TF attacks require to train a neural network before making any prediction. This training phase can take some time, but can be done offline.

In previous work, researchers evaluate attacks with their own data (i.e., collected on their computers). With the implementation we propose, we use real users data to update our attack, without obtaining their consent. This raises an ethical concern. However, the data collected should cause minimal harm to the user since we don't keep all packet sequences but only their corresponding mean feature vectors. As a counter measure, we decided to keep the data on memory only when we need it and delete it when our experiments are finished, i.e. when the server is turned off.

8 Conclusion

In this paper we studied the practical feasibility of large scale WF attacks. We first proposed to use Tor exit nodes to create the WF attack training set, which gives a large number of recent packet sequences to deal with. This is why we also proposed some improvements to existing attacks so that they could update their dataset quickly and make fast predictions. The large-scale attack we propose uses the Average point with TF-MEV attack, which shows during our experiments few errors and better classification times than state-of-the-art attacks.

Our attack is limited by the WF defenses. Indeed, as shown in the TF paper [8], the accuracy of the TF attack would significantly decrease if some defence such as WTF-PAD [4] was used to protect packet sequences.

We have not been able to exploit the packet sequences of real data coming from a Tor exit node yet, it is going to happen in the near future. We will be able to check if, on a large number of real packet sequences, we get results that scale as expected.

Large-scale WF attacks presented in this paper show the capabilities of WF attacks for mass surveillance. This type of tool could affect the privacy of all of us.

References

[1] *Alexa — The Web Information*. www.alexa.com.

[2] *Creme Library*. https://github.com/creme-ml/creme.

[3] Jamie Hayes and George Danezis. "k-fingerprinting: A Robust Scalable Website Fingerprinting Technique". In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 1187–1203. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/hayes.

[4] Marc Juarez et al. "WTF-PAD: Toward an Efficient Website Fingerprinting Defense for Tor". In: (Dec. 2015).

[5] Sattar Hashemi Reyhane Attarian Lida Abdi. "Adaptive Online Website Fingerprinting Attack for Tor Anonymous Network: A Stream-wise Paradigm". In: *Computer Communications*. Department of Computer Science, Engineering, Information Technology, School of Electrical, and Computer Engineering, Shiraz University, Shiraz, Iran, 2019, pp. 74–85. URL: https://doi.org/10.1145/3319535.3354217.

[6] *scikit-learn – Machine Learning in Python*. https://scikit-learn.org/stable/.

[7] Payap Sirinam et al. "Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning". In: *CoRR* abs/1801.02265 (2018). URL: http://arxiv.org/abs/1801.02265.

[8] Payap Sirinam et al. "Triplet Fingerprinting: More Practical and Portable Website Fingerprinting with N-Shot Learning". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom: Association for Computing Machinery, 2019, pp. 1131–1148. URL: https://doi.org/10.1145/3319535.3354217.

[9] *Tor Project*. https://www.torproject.org/.

[10] Tao Wang and Ian Goldberg. "Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks". In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1375–1390. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/wang-tao.