Triplet Fingerprinting: More Practical and Portable Website Fingerprinting with N-shot Learning

Conference Paper · November 2019

DOI: 10.1145/3319535.3354217

4 authors, including:

Payap Sirinam
Rochester Institute of Technology
7 PUBLICATIONS **36** CITATIONS
SEE PROFILE

Mohammad Saidur Rahman
Rochester Institute of Technology
13 PUBLICATIONS **3** CITATIONS
SEE PROFILE

Triplet Fingerprinting: More Practical and Portable Website Fingerprinting with N-shot Learning

Payap Sirinam
Navaminda Kasatriyadhiraj Royal Air Force Academy
Bangkok, Thailand
payap_siri@rtaf.mi.th

Nate Mathews
Rochester Institute of Technology
Rochester, New York
nate.mathews@mail.rit.edu

Mohammad Saidur Rahman
Rochester Institute of Technology
Rochester, New York
saidur.rahman@mail.rit.edu

Matthew Wright
Rochester Institute of Technology
Rochester, New York
matthew.wright@rit.edu

ABSTRACT

Website Fingerprinting (WF) attacks pose a serious threat to users' online privacy, including for users of the Tor anonymity system. By exploiting recent advances in deep learning, WF attacks like Deep Fingerprinting (DF) have reached up to 98% accuracy. The DF attack, however, requires large amounts of training data that needs to be updated regularly, making it less practical for the weaker attacker model typically assumed in WF. Moreover, research on WF attacks has been criticized for not demonstrating attack effectiveness under more realistic and more challenging scenarios. Most research on WF attacks assumes that the testing and training data have similar distributions and are collected from the same type of network at about the same time. In this paper, we examine how an attacker could leverage N-shot learning—a machine learning technique requiring just a few training samples to identify a given class—to reduce the effort of gathering and training with a large WF dataset as well as mitigate the adverse effects of dealing with different network conditions. In particular, we propose a new WF attack called Triplet Fingerprinting (TF) that uses triplet networks for N-shot learning. We evaluate this attack in challenging settings such as where the training and testing data are collected multiple years apart on different networks, and we find that the TF attack remains effective in such settings with 85% accuracy or better. We also show that the TF attack is also effective in the open world and outperforms traditional transfer learning. On top of that, the attack requires only five examples to recognize a website, making it dangerous in a wide variety of scenarios where gathering and training on a complete dataset would be impractical.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; • **Networks → Network privacy and anonymity**.

CCS '19, November 11–15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6747-9/19/11…$15.00
https://doi.org/10.1145/3319535.3354217

KEYWORDS

Tor; privacy; website fingerprinting; deep learning; n-shot learning; triplet networks

ACM Reference Format:
Payap Sirinam, Nate Mathews, Mohammad Saidur Rahman, and Matthew Wright. 2019. Triplet Fingerprinting: More Practical and Portable Website Fingerprinting with N-shot Learning. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3319535.3354217

1 INTRODUCTION

The Tor anonymity system provides privacy to eight million users a day [3, 19], but it has been shown to be vulnerable to a traffic analysis attack called *website fingerprinting (WF)*. WF exploits the fact that the network traffic of each website has its own unique pattern, and these patterns can be learned by a machine learning classifier. The attacker must train the classifier by collecting a large body of network traces between his client and the Tor network from his own visits to websites of interest (*monitored websites*) and other websites that users might visit (*unmonitored websites*). With the trained classifier in hand, the attacker then intercepts traffic in an encrypted connection between the victim and the first Tor node and uses the classifier to determine whether she visited a monitored site and if so, which site she visited. This allows a local and passive network adversary as depicted in Figure 1, such as the victim's Internet service provider (ISP), someone sniffing the wireless connection, or a compromised home router, to link the user with her websites and break the anonymity provided by Tor. When compared to the alternative end-to-end traffic confirmation attack[1], a weak WF attacker needs to be at only the client end of the communication steam.

Previous work in WF attacks [5, 6, 11, 21, 23, 26, 35] demonstrated effective performance in both the *closed-world* setting, in which the user is assumed to only visit sites in the monitored set, and the more realistic *open-world* setting, in which the user might visit any website, whether monitored or not. The state-of-the-art WF attack, *Deep Fingerprinting (DF)*, uses a deep learning classifier to achieve over 98% accuracy in the closed world and over 0.9 for both

[1] An end-to-end confirmation attack allows an attacker to associate a Tor client with their destination by capturing on both ends of the circuit [20, 30, 40], and—unlike the WF attack—is outside Tor's threat model.

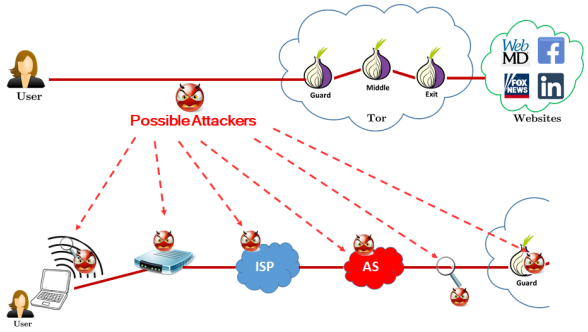


Figure 1: Possible local and passive network adversary that can perform WF attacks.

precision and recall in the open world [29]. Moreover, the DF attack can effectively undermine WTF-PAD, the WF defense that is the main candidate to be deployed in Tor[2] with over 90% accuracy in the closed world and over 0.9 precision in the open world. More recent works offer similar [22] or even better success [7].

While effective, DF requires large amounts of training data that must regularly be updated, and this may not be practical for the weaker attacker model typically assumed in WF. Moreover, there has been criticism that WF attacks may not be effective in realistic conditions [14, 25, 26, 36]. This is because most research in WF attacks makes strong assumptions, such as that the testing and training data have similar distributions and are collected with the same type of network conditions at about the same time. These assumptions may provide unrealistic advantages to attackers or overlook limitations that can substantially reduce the performance of WF in real attack scenarios. In this work, we thus revisit these assumptions and examine approaches an attacker might use to address them. In particular, the key contributions of our work are:

- We identify the following requirements for effective and realistic WF attacks: generalizability to variable testing conditions, low bootstrap time, flexibility in applying to any website, transferability between different scenarios, and high performance.
- We propose a new attack, Triplet Fingerprinting (TF), to satisfy our realistic attack requirements. Our attack uses triplet networks for N-shot learning (NSL) to achieve up to 95% accuracy using only 20 examples per website.
- Furthermore, we investigate a challenging scenario in which the feature extractor is pre-trained with a three-year-old dataset, and we find that TF achieves near 85% accuracy when using only five examples per class.
- We are the first to examine the use of *transfer learning* to improve the bootstrapping of models, but we find that its performance is lacking when compared to triplet networks.
- Finally, we show that TF remains effective in a small open-world setting, achieving approximately 0.9 precision and 0.8 recall when tuned for precision. Performance degrades significantly, however, to 0.3 precision and 0.7 recall when the size of the world is significantly increased.

[2] As of Tor 0.4.0.5, a padding mechanism modeled after WTF-PAD has been used as an implementation of proposal 254 [4].

Overall, we find that the new TF attack provides compelling properties that can help overcome or mitigate the adverse effects of more realistic and more challenging scenarios that an attacker may face. As the attack enables an adversary with low computing resources and limited time to perform WF, it further shows the seriousness of WF attacks in undermining the anonymity of Tor. Finally, the findings set up a new direction of research to further study how to apply NSL for more realistic WF attacks and how to effectively counter TF and other NSL-based attacks.

2 BACKGROUND AND RELATED WORK

2.1 WF Attacks

Many researchers have examined WF since Herrmann et al. first evaluated the attack against Tor [13] in 2009. Until recently, WF attacks used traditional machine learning classifiers, in which the attacker needs to perform manual feature engineering to select a set of features to represent a site. This type of attack has been shown to be effective with over 90% accuracy on 90 training instances, and includes works such as k-NN [35], CUMUL [23] and k-FP [11].

The emergence of deep learning (DL) in the last few years has motivated researchers to apply DL to improve WF attacks. In 2016, Abe and Goto explored the use of a Stacked Denoising Autoencoder (SDAE) in WF with moderate success [5].Following this, Rimmer et al. demonstrated more broadly that DL can be used to automate the feature engineering process. In their evaluations, they compared the performance of several deep-learning models such as Convolutional Neural Network (CNN), Long-Short Term Memory (LSTM), and SDAE using a large dataset of 900 sites with 2,500 instances each. They find that SDAE provides the best results, with 96.3% accuracy in the closed world and 71.3% TPR and 3.4% FPR in the open world.

Sirinam et al. proposed the Deep Fingerprinting (DF) attack [29], a CNN-based WF attack that incorporates sophisticated properties of recent CNNs used in computer vision. The DF attack can achieve 98.3% accuracy in the closed world and 99% precision with 94% recall in the open world. Moreover, the DF attack is the first to effectively undermine the WTF-PAD defense [15], which is considered the primary candidate for deployment in Tor. Sirinam et al. showed that the DF attack can achieve 90.7% closed-world accuracy against the WTF-PAD defense.

More recently, Oh et al. [22] explored the use of unsupervised deep learning and highlight the utility of their model for feature extraction separately from the classification task. Their results are not better than those for DF. Bhat et al. [7], on the other hand, developed Var-CNN, a more sophisticated CNN than the DF approach based on techniques that are specific to the WF problem and help it achieve results that are the new WF state-of-the-art. More importantly with respect to our research, it works well in relatively low-data settings. With 40 traces per site, it gets about 92.4% accuracy in the closed world. In a large open world with 40 traces per site, it gets 77% TPR, though the FPR, precision, and recall are not reported. Unfortunately, these results were presented while the current paper was being written, so we are not able to provide detailed comparisons with Var-CNN in low-data settings in this work. Thus, our work mainly compares with the DF attack.

While these WF attacks achieve high performance in lab conditions, several research groups have criticized such environments as

unrealistic [14, 25, 26, 36]. WF research attempts to model the real world by performing experiments under a set of assumptions. Over time, the community has revised and improved these assumptions to be more realistic, but some of the key assumptions have not been considered in the design of attacks. In this work, we examine how an adversary can engineer attacks that are more applicable in a real-world environment.

2.2 WF Attack Assumptions

In this section, we summarize and categorize the current assumptions that have been made in the WF literature. This allows us to identify which attacker constraints have been appropriately evaluated and which have not.

2.2.1 Closed- vs Open-world Scenario. WF attacks are evaluated under two possible scenarios: closed world and open world. The closed world assumes that there are only k websites that the client can visit, where k is far smaller than the number of websites available in the real world. Despite being criticized as unrealistic [14], closed-world evaluation is still used as a metric to evaluate the quality of attacks and feature sets. Subsequent work also considers the open-world scenario to measure attack efficacy in a realistic setting. Here, we examine two additional constraints when designing an open-world experiment:

- *Size of the open world.* Researchers have increased the number of unmonitored websites in open-world datasets to evaluate the ability of the WF classifier to distinguish between monitored and unmonitored sites [11, 23, 26, 29, 35]. The size of the unmonitored set has reached up to 400,000 websites in the dataset of Rimmer et al. [26]. While even larger sets would be more representative, given that the victim could be visiting any page on any site in the world, there are diminishing returns above a certain size. Further, it is likely that sites have widely varying popularity for Tor users, so capturing the extended tail of the distribution may not be meaningful in practice. In this paper, we do not attempt to address the question of the most appropriate size of the unmonitored set.
- *Open-world evaluation model.* There are two models used to evaluate the performance of WF classifiers in open world: the *Standard* model and the *AWF* model [26]. Under the Standard model, samples from the unmonitored set are included in the training data as an additional label. Researchers assume that doing so will help the classifier to better distinguish between the monitored and unmonitored websites. This model was used by the majority of prior works [11, 23, 29, 35]. On the other hand, the AWF model does not include unmonitored websites in the training data set. The classifier instead uses a confidence threshold based on the cross-entropy loss function to identify unmonitored sites. Rimmer et al. [26] argue that even if the attacker may gain benefit from including the unmonitored websites, the size of the unmonitored set is still not representative of the actual size of the world of websites. The choice of which model is best is an interesting point to consider, but we stick with the more popular Standard model for this paper and leave the question for future work.

Table 1: Impact on attack accuracy when training and testing with different TBB versions. Data from Juarez at al. [14].

TBB Version	2.4.7 (Test)	3.5 (Test)	3.5.2.1 (Test)
2.4.7 (Train)	62.70 ± 2.80	29.93 ± 2.54	12.30 ± 1.47
3.5 (Train)	16.28 ± 4.51	76.38 ± 4.97	72.43 ± 3.22
3.5.2.1 (Train)	6.51 ± 1.15	66.75 ± 3.68	79.58 ± 2.45

2.2.2 Users' Browsing Behavior. Most prior work [11, 23, 26, 29, 35] assumes that Tor clients follow a rather specific behavior: they use Tor to browse websites sequentially, and they use only a single tab at a time so that website visits do not overlap. This is, of course, not representative of real-world behavior of a Tor client. As Tor connections are slow, it is likely that clients will open several browser tabs and visit sites concurrently [1, 33, 37]. The effects of multi-tab browsing have been explored in prior work [14, 36], and we do not further examine them in this paper.

2.2.3 Traffic Parsing and Background Traffic. In WF attacks, we assume that the attacker can collect all the traffic generated by a site and can effectively distinguish this traffic from other traffic connections [14]. This assumption is guaranteed to be true only when the attacker performs the attack at the guard node, which allows him to extract the specific traffic by using the Tor circuit ID. In the case that the attacker performs the attack as an eavesdropper between the client and the guard node, all Tor traffic is multiplexed in the TLS connection. Recent work has developed techniques to effectively discriminate Tor traffic from multiplexed TLS traffic and split it into corresponding encrypted connections to each website [36]. Thus, this assumption has been already handled and is not the focus of our study.

3 ATTACKER GOALS

In this section, we identify elements of WF attacks that may be improved to better suit realistic adversaries. Prior work on attack design has largely discounted these requirements.

3.1 Generalizability

Previous research assumes that a WF attacker can train his classifier under the same conditions as the victim, effectively making it a *targeted* attack. Given this assumption, the attacker can replicate the victim's network conditions, Tor-browser-bundle (TBB) version, and settings, and these can impact the attack's performance [14]. Alternatively, the attacker may be interested to attack more than one victim, i.e. to perform a *untargeted* attack. In this case, the attacker should be prepared for users with different types of network connections, TBB versions, and settings. This constraint makes WF more difficult as the adversary's classifier must remain effective even when using such a diverse data set. Juarez et al. show [14] that if the attacker had trained the WF classifier based on one TBB version and tested with another, the accuracy of the attack drops drastically as shown in Table 1.

Recent WF attack research [11, 23, 26, 29, 35] assumes that the attacker performs a targeted attack by crawling a single data set using similar machines under the same network conditions. This data set is then used in both the training and testing phases, which

Table 2: Bootstrap time to create a WF classifier (AWF [26]). The crawling rate is approx. 2,000 instances/day/computer and 500,000 instances are required for training.

No. of PCs	Crawling Time	Training Time (w/GPU)	Bootstrap Time
1	250 days	<1 hour	250 days
4	63 days	<1 hour	63 days
8	32 days	<1 hour	32 days
12	21 days	<1 hour	21 days
24	11 days	<1 hour	11 days

gives the attacker an unrealistic advantage for any untargeted attack, as the distribution of traces would be more heterogeneous in the real world. In this work, we investigate how a WF attack can be crafted to avoid this assumption.

3.2 Bootstrap Time

To perform WF attacks, the attacker needs to train the classifier to predict the unknown traffic captured from the client. This does not, however, capture the *bootstrap time*, the total amount of time required for the attacker to produce a *ready-to-use* classifier. This time includes both the time required to crawl a training data set and to train the classifier. It is important to note that the traffic traces of websites regularly change over time due to many factors, such as changes in the website, changes in network conditions, and changes in Tor. Thus, the attacker needs to collect new network traffic frequently to avoid significant mismatches between testing and training data.

To the best of our knowledge, there is no WF research that comprehensively considers the bootstrap time. Prior work makes the assumption that the dataset used to test the classifier is from the same distribution that is used for training. However, in a realistic scenario, the gap time between the training and testing phases may be large enough to cause potential data mismatch issues that would negatively affect performance. This concern has been studied in prior work. Juarez et al. found that attack accuracy for k-NN dropped from 80% to 30% within 10 days [14], while Rimmer et al. found a drop in accuracy from 95% to 81% after 28 days using an SDAE classifier [26]. Therefore, we can infer that the trained classifier will only remain effective for a few weeks at best after the classifier is initially trained.

The need to frequently re-train the classifier raises a question in regards to the computing resources and time required for the attacker to collect the new data. The longer the time used to collect the new data, the higher the chance that the dataset will be stale by the time the attack is deployed. Table 2 shows the time required for crawling the data (with typical PCs on a fast network connection) used for training the classifier for the AWF model [26]. Even when the attacker uses two dozen computers for crawling, nearly two weeks are needed to collect this large data set.

Wang and Goldberg examined the issues of bootstrap time and data freshness for the k-NN attack [36]. They find that k-NN trained on as few as 31 instances per site is nearly as effective as using 85 instances per site, with 0.77 TPR and 0.003 FPR. Bhat et al. [7] experiment with Var-CNN using as few as 40 traces and also show good results. For five instances per site, however, Wang and Goldberg report a TPR of just 0.253. Additionally, the attacker would still need to collect thousands of traces to get 31-40 instances per site with a sufficiently large unmonitored set.

Our attack uses as little as five traces *in total* given a pre-trained model. To address data freshness, Wang and Goldberg propose a technique to update a trained k-NN classifier. This technique needs to be run continuously, however, to maintain the freshness of data. If the attacker obtains an older dataset, it will be too stale to help in their approach. Our model, on the other hand, can use a *years-old dataset* to build the pre-trained model.

3.3 Flexibility & Transferability

A WF attack is a traditional classification problem in which the attacker uses a fixed number of labels for training the classifier. During the training process, the classifier is trained by learning to locally map the input data to the given website. These following steps briefly describe the training and prediction process:

- The attacker first determines the set of k monitored sites and labels them as $s_1, s_2, \ldots s_k$.
- He then gathers T training instances for each monitored site and another U training instances for the unmonitored set.
- The attacker trains his classifier with the $k \times T + U$ collected training traces with their corresponding labels.
- He eavesdrops on the victim to capture an unlabeled trace.
- In the prediction phase, the attacker uses the trained classifier to predict the label of the victim's trace.

Note that the possible predicted class is then limited to one of the k websites that were previously used for training or to the unmonitored set. Thus, the attacker is subject to an additional constraint. Whenever the attacker wishes to add or remove a website from his monitored set, he must re-train his classifier. This is yet another issue which increases the time and resource requirements necessary to perform the attack. In this work, we study how techniques such as N-shot learning and transfer learning can be used to improve WF attacks. It is important to note that resolving the flexibility and transferability issues can directly ameliorate the bootstrap time and generalizability issues, since the time required to collect data is reduced and more varied data set may be collected.

3.4 Goals for Improvement

We have described three areas of improvements of WF attacks, including generalizability, bootstrap time, and flexibility and transferability issues. To improve the performance of WF attacks, we have established the following key goals to address each issue.

Generalizability. The WF classifier should be robust to the data mismatch issues that occur as a result of 1) staleness of training data and 2) heterogeneous distributions of training and testing data.

Bootstrap time. The WF classifier, which is trained on one dataset, should remain effective against traces collected later. If it needs to be re-trained, the amount of training data required should be small to reduce the effort needed for data collection.

Flexibility & Transferability. The WF classifier should enable the attacker to flexibly add new sites to the monitored set or use

a completely new monitored set with only modest effort in data collection and training.

Attack Performance. After achieving the aforementioned goals, a robust classifier must of course still achieve a high level of accuracy.

A classifier that is able to achieve these goals is much more dangerous to the privacy of Tor users than one that requires the attacker to have significant computing resources for frequently gathering fresh training data specifically targeting each victim's circumstances. In the next section, we describe a technique that leverages N-shot learning to meet these requirements.

4 N-SHOT LEARNING

DL has shown to be effective in many domains of applications such as image recognition, speech recognition, and WF attacks [8, 12, 16, 18, 28, 29, 31]. However, traditional supervised DL algorithms normally require 1) a large number of labeled examples used for training the classifier, and 2) distributions of training and testing datasets that are matched or at least similar. Moreover, a model can only make predictions for the set of classes on which it was trained.

This style of learning contrasts with what we normally think of as true intelligence. For example, a person can recognize the face of someone after only seeing them a few times, and this ability scales to thousands of different faces. The DL models currently in use for WF are unable to do this. This is a key challenge in DL: How can we build a model that can rapidly learn from very little data? This challenge has motivated the development of the *N-shot Learning* (NSL) technique [9, 34].

4.1 NSL Implementation

NSL is a recently developed ML procedure that allows a model to accurately classify samples based on only a few training examples. More precisely, NSL requires only a small number N of examples for every given class. So, when N=5 (called 5-Shot Learning), the classifier learns from a training dataset that contains only five samples for every class. NSL has been broadly implemented in face recognition [24, 27], where it is a compelling approach due to constraints inherent in the task:

Limited training data. The classifier used to perform the face recognition task cannot expect a rich dataset of training data. For example, if we want to design a face recognition system for use at a company, the system cannot require hundreds of photos from each employee, as that would be impractical to implement.

Ability to update class labels. In most uses of these classifiers, it is expected that class labels will need to be added or removed from the system frequently. For example, if there are new employees, the classifier should still be able to run without downtime for re-training the classifier.

NSL is able to address both of these constraints by modifying the learning process. We summarize the key differences between the NSL and traditional supervised learning as the following:

Learning goal. Traditional supervised learning mainly focuses on training the classifier to learn and locally map the input to its corresponding class. In contast, NSL models are trained to learn how to distinguish between different objects regardless of the previously trained classes.

Prediction output. Traditional supervised learning aims to simply predict a certain class within the set of training examples. By contrast, the model in NSL is treated as a feature extractor to generate the embedded vectors of inputs from the learned model. These embedded vectors are used to measure similarity, and the expected prediction output is to decide whether or not these inputs are in the same class.

Transferability and flexibility. The transferability of the model enables the practitioner to use models pre-trained by others and make small changes to it. It is important to note that a rich dataset is still necessary to initially train the NSL model. However, the NSL model is used as a feature extractor without being locally bounded to a set of classes. Thus, it can more readily generalize to new classes. Moreover, NSL allows the practitioner to flexibly adopt a pre-trained model from others who have more computing resources and larger training datasets.

Number of learning examples. After the underlying model used by NSL is trained, only a small number of examples are required for each class to generate an embedded vector. The embedded vectors are then used to train the final classifier to classify the given inputs into their corresponding classes.

An early implementation of NSL used k-Nearest Neighbours (k-NN) to directly measure the similarity between two different samples, but this showed poor results due to being overly sensitive to minor variations in raw data. To mitigate this problem, the model needs to be capable of effectively extracting representative features that are robust to variation before distance between samples is measured. Koch et al. demonstrated that using deep learning for feature extraction is an effective solution to this problem [17]. There are two deep embedded networks that have seen use in NSL: *Siamese Networks* [32] and *Triplet Networks* [27]. Siamese networks are conceptually based on similarity learning, in which we measure how similar two comparable objects are. Triplet networks have been shown to be more effective than Siamese networks [24, 27], and we confirmed this for the WF problem in a preliminary study.

4.2 Triplet Networks

Triplet networks [24, 27] contain parallel and identical sub-networks sharing the same weights and hyperparamenters as shown in Figure 2. Three different inputs called *triplets* are used to train the networks. The triplets are randomly sampled from the training data to create an array containing the vectors of three different input examples: *Anchor (A)*, *Positive (P)*, and *Negative (N)*. Each input is individually fed to their corresponding sub-network during the training phase. To explain the differences between A, P, and N, let us craft a toy example. Let us say we have a dataset that contains traffic examples from three different websites—*wikipedia.org*, *gmail.com* and *amazon.com*—and each website has five examples. Then sampling and generating the triplets used to train the network proceeds as:

- **Anchor Input (A)**: The anchor input is the example used as the main reference—e.g. the first example from *wikipedia.org*.

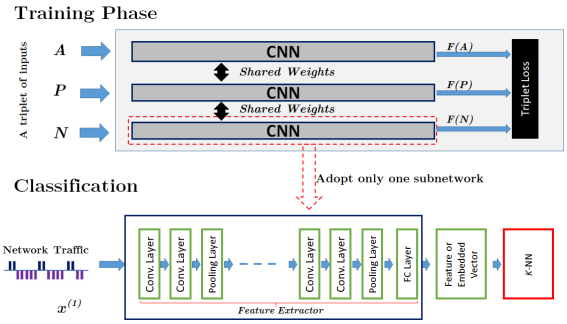


Figure 2: The training phase of triplet networks and their implementation for a classification task.

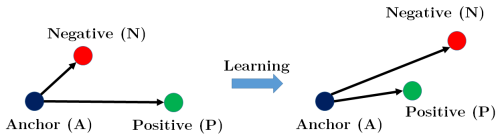


Figure 3: The learning process while training the model: The triplet loss minimizes the distance of examples from the same website and maximizes the distance for examples from different websites.

- **Positive Input (P)**: The positive input is chosen from the remaining examples of the anchor's class—e.g. the second example from *wikipedia.org*.
- **Negative Input (N)**: The negative input is sampled from any class that is not the anchor—e.g. any of network traffic's examples from *gmail.com*, or *amazon.com*.

The formation of triplets is used to allow the triplet networks to learn how to differentiate between different objects. In the training phase as shown in Figure 2, we feed the triplets to the sub-networks; anchor input A for the first sub-network, positive input P for the second sub-network, and negative input N for the third sub-network. Then the networks measure the similarity based on distance metric D of anchor A with both P—$D(A, P)$—and N—$D(A, N)$—using their embedded vectors.

To achieve the model's learning goal, the triplet network is trained to produce embeddings for A, P, and N such that the distance $D(A, P)$ becomes less than the distance $D(A, N)$. In other words, we expect the network traffic traces from the same website to be similar to each other (small distance between A and P) and network traffic from different websites to be dissimilar (large distance between A and N). Figure 3 illustrates the expected learning process of the model, showing that during training, the model is learning to move A and P closer to each other and separating A and N farther apart in the embedding space.

Once the model training process is accomplished, we only use one of the sub-networks as the feature extractor to perform the classification task as shown in Figure 2.

Table 3: Hyperparameter selection for the TF triplet network (feature extractor)

Parameters	Search Space	Selected Value
Base Model	GoogleNet, RestNet, Xception, DF	DF
Similarity Metrics	Euclidean, Cosine	Cosine
Mining Strategy	Random, Hard-Negative, Semi-Hard-Negative	Semi-Hard-Negative
Margin	[0.0 .. 0.5]	0.1
Optimizer	SGD, Adam, Adamax, RMProp	SGD
Batch Size	[32 ... 256]	64, 128
Embedded Vector's Size	[32 ... 256]	64

5 TRIPLET NETWORKS IN WF

We propose the *Triplet Fingerprinting (TF)* attack to perform WF attacks under more realistic and more challenging scenarios by using NSL with triplet networks. Our implementation of the TF attack uses the Python DL libraries *Keras* [2] as the front-end and *Tensorflow* as the back-end.

5.1 Hyperparameter Tuning

To develop TF, we have followed the implementation guidelines and techniques from the previous work on applying NSL [24, 27, 32]. Moreover, we perform hyperparameter tuning on the DL model to identify the best model and hyperparameters. We follow the extensive candidates search method [29] to evaluate and select the final value for each hyperparameter. Of note:

Base Model. The base model refers to the CNN model that is used as the sub-network in the triplet networks as shown in Figure 2. We test with the DF model, as well as using techniques from Sriniam et al. [29] to adapt previously-proposed DL models used in image recognition, namely GoogleNet [31], RestNet [12] and Xception [8], for the WF problem. We find that the DF model performs better than the other candidates with significantly less training time, so we selected it for our base model.

Distance Metrics. We evaluate two traditional metrics: *Euclidean distance* and *Cosine distance*. We find that Cosine distance provides better results. Cosine distance has meaningful semantics for ranking similarity based on mutual object frequency, whereas the Euclidean distance does not. Ranking similarity based on mutual objects is similar to finding the common bursts of traffic patterns, which is one of the meaningful features in WF attacks.

Mining Strategy. *Mining* is the process of identifying triplets to use for training the feature extractor. Selecting random examples P from the same class as the anchor A is generally fine, but most negative examples N are too different from A to help us learn how to extract useful features. *Hard negative* examples are examples N that are more likely to be close to A, such that the feature extractor will need to identify features that are useful for discrimination to get a larger distance for $D(A, N)$ than $D(A, P)$. We evaluate three different mining strategies: *Random*, *Hard-Negative* and *Semi-Hard-Negative*. The results demonstrate that Semi-Hard-Negative provides the best results, which is consistent with prior work [10, 27].

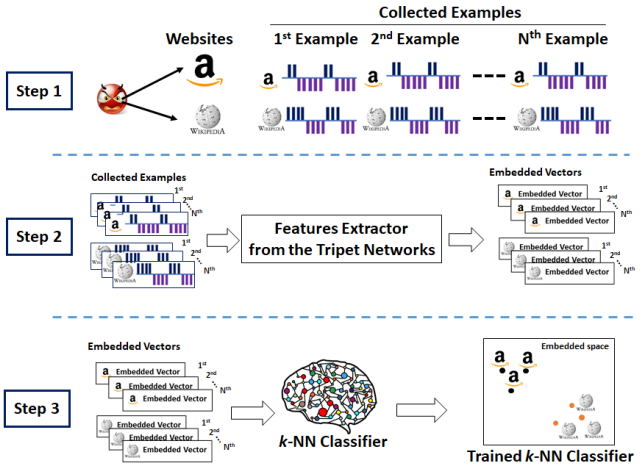


Figure 4: NSL N-Training

Table 3 shows a summary of the hyperparameter tuning process, including the set of main parameters that we evaluate, the range we searched, and the final selected value for each parameter. We also provide additional details of the hyperparameter tuning process for all hyperparameters along with intuitive explanations and the preliminary results used for each selection in Appendix D.

5.2 TF Attack Implementation

We now explain the step-by-step implementation of the TF attack, including how to train the triplet networks (pre-training phase) to generate the pre-trained model and how to use the model to perform classification (attack phase).

5.2.1 Pre-Training Phase. The attacker creates the triplet networks and uses the DF model as the sub-network along with fine-tuned hyperparameters. It is important to note that we replace the softmax layer in the DF model with the new FC layer used as the embedded vector. In every epoch of training, the system feeds the batch of triplet inputs generated from semi-hard-negative mining into the model, as shown in Figure 2. The outcome of the pre-training phase is the feature extractor that is later used to help perform the classification task.

5.2.2 Attack Phase. The attacker performs two steps during the attack phase: N-training and classification. We use the term **N-training** for the process of training his WF classifier from the N embedded vectors generated from the triplet network's feature extractor. This helps prevent confusion with the **pre-training phase**, which refers to the process of training the triplet networks to generate the feature extractor. N-training and classification proceed as follows:

N-training. As shown in Figure 4:

- **Step 1**: The attacker collects N examples from each of the monitored websites (e.g. five examples for each site in 5-shot learning).
- **Step 2**: The collected traffic examples are fed to the feature extractor (triplet networks' pre-trained model) to generate corresponding embedded vectors for each site.

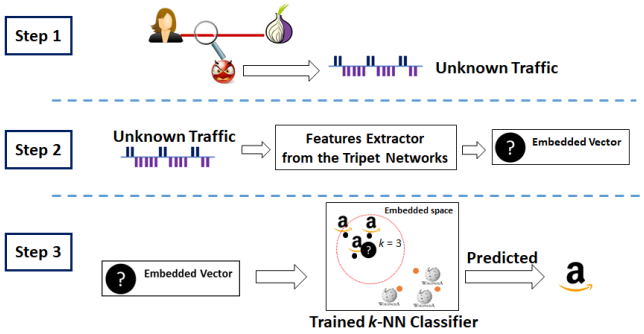


Figure 5: NSL Classification

- **Step 3**: The embedded vectors are used to train a k-NN classifier. Note that the attacker can use any type of classifier such SVM, MLP, CNN, etc. A preliminary study indicated that k-NN significantly outperforms other classifiers for this purpose.

Classification. Using the trained classifier to perform the WF attack proceeds as shown in Figure 5:

- **Step 1**: The attacker captures the unknown network traffic from the user.
- **Step 2**: The unknown traffic is fed into the same feature extractor used during the N-training process to generate the embedded vectors for the unknown traffic.
- **Step 3**: The attacker uses his trained k-NN classifier to predict a label for the embedded vector of the unknown traffic sample.

6 EXPERIMENTAL EVALUATIONS

In this section, we design a series of experimental evaluations to investigate the performance of the TF attack under different scenarios with respect to the desired improvements, including generalizability, bootstrap time, flexibility, and transferability.

6.1 Dataset

We perform our experiments using datasets provided by other researchers and used in the previous literature. We label these datasets as follows:

Wang dataset [35]. This dataset contains a set of monitored websites selected from a list of sites blocked in China, the UK, and Saudi Arabia. The unmonitored websites were chosen from the Alexa Top sites.[3] The dataset was collected in 2013 using TBB version 3.X. We label the two subsets as:

- **Wang100**: The set of 100 monitored websites, where each website has 90 examples.
- **Wang9000**: The set of 9,000 unmonitored websites, where each website has one example.

AWF dataset [26]. This dataset includes monitored websites from the 1,200 Alexa Top sites, and unmonitored websites from the 400,000 Alexa Top sites. The dataset was collected in 2016 using TBB

[3] https://www.alexa.com/topsites

Table 4: The performance of prior attacks for NSL (Accuracy)

Type of Experiment	Number of N Example(s)				
	1	5	10	15	20
CUMUL [23]	42.1 ± 5.5	72.2 ± 1.7	79.7 ± 1.4	83.3 ± 2.0	85.9 ± 0.6
k-FP [11]	36.3 ± 1.6	79.3 ± 1.0	83.9 ± 1.0	85.9 ± 0.6	87.5 ± 0.8

version 6.X. We categorize the AWF dataset into several different sets:

- **AWF100**: The set of the first 100 monitored websites, where each website has 2,500 examples.
- **AWF775**: The set of the other 775 monitored websites, where each website has 2,500 examples .
- **AWF9000**: A set of 9,000 unmonitored websites, where each website has one example.
- **AWFλK**: A subset with $\lambda \times 1000$ of the 400,000 unmonitored websites, where each website has one example.

DF dataset [29]. This dataset consists of both monitored and unmonitored websites crawled from the Alexa Top sites. As with the AWF dataset, this dataset was collected in 2016 using TBB version 6.X. We categorize the DF dataset into two sets:

- **DF95**: The set of 95 monitored websites, where each website has 1,000 examples.
- **DF9000**: The set of 9,000 unmonitored websites, where each website has one example.

We choose these three datasets to support the different purposes of our experiments. The intuitive explanation behind the selection of each dataset will be later described in each experimental setup.

Data Representation. We follow the data representation used by recent work in WF using DL [26, 29, 35]. The data used for training and testing the model consists of network traffic examples from various sources of data as mentioned above. All examples are converted to sequences in which we ignore packet size and timestamps and only store the traffic direction of each packet, with +1 and -1 representing outgoing and incoming packets, respectively. The sequences are trimmed or padded with 0's as need to reach a fixed length of 5,000 packets. Thus, the input forms a 1-D array of [n x 5000], where n is the total number of examples fed into the model.

6.2 Statistical soundness

We run the experimental testing 10 times and find the mean and standard deviation to report the final performance of the attack. Furthermore, the network traffic examples used for N-training the classifier and for testing the classifier are randomly shuffled and sampled at every round of the evaluation to ensure that the results are not evaluated from only specific data points.

6.3 Prior work baseline

To begin, we have reevaluated prior attacks CUMUL and k-FP under the training sample restrictions of NSL.[4] For these experiments, we split the AWF100 dataset into testing and training portions. The number of samples used in the training portion is varied throughout the experiments. The results for N=[1, 5, 10, 15, 20] training samples per class can be seen in Table 4.

[4]We were unable to accurately reproduce the results of Yan and Kaur [38] and consequently we do not include the *wfin* attack in our baseline experiments.

Table 5: The performance of WF attacks: Similar but mutually exclusive datasets (Accuracy)

Type of Experiment	Embedded Vectors	Number of examples N				
		1	5	10	15	20
Disjointed Websites	N-ALL	79.4 ± 1.6	90.9 ± 0.7	93.1 ± 0.2	93.3 ± 0.3	93.9 ± 0.2
	N-MEV		92.2 ± 0.6	93.9 ± 0.2	94.4 ± 0.3	94.5 ± 0.2

6.4 Similar but mutually exclusive datasets

The first experiment evaluates the attack scenario in which the attacker pre-trains the feature extractor on one dataset and performs classification on a different dataset with different classes (Disjointed websites). More precisely, the websites' URLs used during the pre-training phase and the attack phase are mutually exclusive. In this scenario, the training and testing datasets have a similar distribution in that they are both collected with from the same period of time (2016) using the same version of TBB (6.X).

Experimental setting. We train the feature extractor by using the AWF775 dataset and test classification performance on AWF100. During the training phase, we randomly sampled 25 examples for each website in the AWF775 dataset using the semi-hard-negative mining strategy to formulate 232,500 triplets.

During the attack phase, we use 90 randomly-sampled examples for each website from the AWF100 dataset. We separate each site's examples into two chunks, with 20 examples for the first chunk and 70 examples for the second chunk. The examples in the first chunk are reserved to evaluate the classification performance on the $N = 1, 5, 10, 15, 20$ examples that are collected by the attacker to N-train the k-NN classifier. The other 70 examples are used as testing data to evaluate the performance of the attack from the trained k-NN classifier. Note that, we will apply these basic experimental settings for the rest of the experiments in this paper.

N-ALL vs N-MEV. The original implementation of N-shot learning classification uses N embedded examples to N-train the k-NN classifiers as mentioned in Section 5.2. We call this representation *N-ALL*. We propose a new approach to improve the performance of the k-NN classifier by modifying the input representation. Instead of using N examples for each website, we calculate the mean of all N examples to generate a Mean Embedded Vector (MEV) used to train the classifier.

We evaluate the classification performance that results from using 1) the *N-ALL* representation in which all N embedded examples for each website are fed to train the model, and 2) our proposed *N-MEV* representation, in which the embedded vector is generated from the mean of N embedded examples for each websites.

Results: Table 5 shows the performance of WF attacks on mutually exclusive training and testing datasets with different values of N. Overall, the N-MEV vectors consistently provide better performance than N-ALL in term of the accuracy of the attack. We believe that the average of vectors in N-MEV helps reduce the noise between embedded vectors of the same class. Therefore, we will mainly use the N-MEV representation for the following experimental evaluations.

Table 6: The impact of including different portions of the datasets during training (Accuracy)

Type of Experiment	Number of N Example(s)				
	1	5	10	15	20
Disjointed Websites	79.4 ± 1.6	92.2 ± 0.6	93.9 ± 0.2	94.4 ± 0.3	94.5 ± 0.2
25% Inclusion	81.2 ± 1.3	92.9 ± 0.6	94.3 ± 0.7	94.7 ± 0.5	94.7 ± 0.3
50% Inclusion	79.6 ± 1.9	92.7 ± 0.8	94.1 ± 0.9	94.7 ± 0.7	95.0 ± 0.5
75% Inclusion	79.7 ± 1.7	93.0 ± 1.4	94.2 ± 0.9	94.5 ± 1.1	95.0 ± 0.8
100% Inclusion	80.6 ± 2.3	93.4 ± 0.9	94.6 ± 0.7	94.7 ± 0.8	95.0 ± 0.9

The results also show that the accuracy of the attack could reach to almost 80% with only one example (1-shot learning). 5-shot learning impressively attains 92% accuracy. We observe that the accuracy starts leveling off after $N \geq 15$ at 94%.

TF Goals. Based on these results, we summarize how the TF attack can improve the performance of WF attacks:

- **Flexibility & transferability**: The results show that even if the classifier is trained on one dataset and tested on a different dataset in term of their websites' labels, the performance remains effective. Therefore, the attacker can directly perform WF attacks without worrying about whether or not the websites that he would like to monitor were included during the pre-training process.
- **Bootstrap time**: The attacker only needs to collect five examples per website, taking ~5 minutes per website, to N-train the classifier. This supports cooperation among attackers, in which one attacker with more time to collect data can periodically train the feature extractor, while other attackers only need to collect data for their sites of interest close to the time of use.
- **Performance of attack**: The results demonstrate that the TF attack can still remain effective with over 90% accuracy.

Overlaps between Pre-Training and Attack Phases. The attacker may believe that including some samples of the same class in both the pre-training and N-training phases will improve performance (ie. the pre-training and N-training datasets are not disjoint). This would allow the triplet networks to train and test with a partial set of websites that the model has seen before. As an example, an attacker may believe that www.bbc.com is commonly selected as a monitored site by other attackers who use his model, so he decides to include examples of www.bbc.com during the training phase. It is then interesting to evaluate whether or not including this site has improved the model's ability to identify new traces from this site. To test this, we perform experimental evaluations to compare the case of disjointed datasets with different percentages of inclusion. The inclusion rates are ranged from from 25% to 100%. The results of these experiments are shown in Table 6.

We find that allowing inclusion between our training sets does not provide noticeable improvement in attack performance. A reason for this may be related to the fact that DL models that use softmax classification learn to locally map the given input to their corresponding class. This leads to overfitting and a more rigid model. By contrast, NSL with triplet networks has the model learn to differentiate the given pair of inputs (similar or dissimilar) without locally mapping to the particular website's label to be assigned. This allows the model to more effectively learn on small numbers of samples. This is the compelling property of NSL which allows us to achieve the goals of flexibility and transferability.

Table 7: The performance of WF attacks with different distributions of training and testing datasets (Accuracy)

Type of Experiment	Number of N Example(s)				
	1	5	10	15	20
Different Distributions	73.1 ± 1.8	84.5 ± 0.4	86.2 ± 0.4	86.6 ± 0.3	87.0 ± 0.3

Furthermore, it is interesting to compare the performance of the TF attack with previously-proposed WF attacks using hand-crafted features, as these attacks have been shown to be effective with less training data than DL attacks. For this purpose, the TF attack trained with 100% inclusion is most appropriate for comparison against the baseline attacks. We find that the TF attack is clearly superior to the baselines when small sample counts are used—e.g. CUMUL and k-FP achieve 42.1% and 36.3% accuracy respectively when $N = 1$ (1-shot learning) whereas TF attack reaches 80.6% accuracy. Although the accuracy improves significantly as N increases for both CUMUL and *k-FP*, the TF attack still significantly outperforms the baselines in all settings. This confirms that the TF attack is distinctive in its ability to achieve high WF performance in low traffic example settings.

6.5 WF attacks with different data distributions

Next, we evaluate the performance of the WF attack under the scenario in which the pre-training and classification datasets are collected at different times with different Tor Browser Bundles (TBB), leading to the data having different distributions.

Experimental setting. We use the same triplet model from the first experiment trained with the AWF775 dataset for feature extraction. To N-train the k-NN classifier, however, we use the Wang100 dataset, which was collected three years prior to the collection of AWF775 using a much older TBB. The experimental setting is designed to evaluate the performance of the attack in which the feature extractor is trained on substantially different data from what the attacker is targeting.[5] To verify that the two datasets are significantly different, we analyze them using Cosine similarity in Appendix A and conclude that these datasets are very likely to be mismatched.

In these experiments, we decided to pre-train TF on a dataset collected in 2016 and test on a dataset collected in 2013 primarily due to the larger variety of websites in the 2016 dataset. Triplet networks learn to classify by identifying the differences between classes, so it is important that a large number of classes are contained in the data that is used to pre-train the feature extractor. Since the objective of this experiment is to evaluate if TF can mitigate the adverse effects of data mismatch between the training phases, we believe the order of the timing does not affect the validity of our evaluations.

Results. As we see in Table 7, the results show that the TF attack remains fairly effective and achieves almost 85% with 5-shot learning. The accuracy gradually increases up to 87% with 20-shot learning.

[5]The attacker must still capture a small dataset of N fresh representative samples for each monitored website.

Table 8: *Transfer Learning*: Similar but mutually exclusive datasets (Accuracy)

Approach	Number of Example(s) N				
	1	5	10	15	20
Traditional	27.9 ± 5.0	87.6 ± 0.4	93.4 ± 0.2	95.2 ± 0.1	95.1 ± 0.1
TF	79.2 ± 1.3	92.2 ± 0.6	93.9 ± 0.2	94.4 ± 0.3	94.5 ± 0.2

TF Goals. These results demonstrate that the TF attack achieves another one of our goals:

- **Generalizability**: The results demonstrate that the feature extractor can be trained on traffic traces having one distribution and used in an attack on traffic traces with a different distribution. Thus, a WF attack using NSL allows the attacker to adopt a feature extractor trained on older data and still perform WF attacks with respectable accuracy.

6.6 Transfer Learning vs Triplet Networks

Transfer learning [39] is a machine learning technique in which a model trained on one task can be effectively re-used on another. This technique has been shown to be effective in many domains. The intuition behind the effectiveness of the transfer learning results from the way that deep learning learns features representations. In computer vision, for example, a deep learning model learns lower-level features such as edges in its earlier layers and higher-level features such as parts of objects in its deeper layers. This hierarchical learning allows users to directly transfer the knowledge of learned features from the early layers and only fine-tune the deeper layers to fit the model for their tasks. Thus, the user does not need to re-train the model from scratch.

These properties make transfer learning appear attractive to address the same issues in WF attacks as the triplet networks that we apply, and so we briefly investigate how well they compare.

To clearly distinguish between the aforementioned transfer learning and the TF attack, we will use the term *traditional* to describe the general transfer learning technique.

Experimental setting: To pre-train the model, we use the DF architecture trained with the AWF775 dataset. We follow the recommendations to re-train the model by freezing k early layers out of the n total layers during the re-training process[6]. We test with different values of k to maximize the accuracy of the attack, and we find that freezing all layers except the last FC layer with softmax provides the highest accuracy. Thus, we use the setting where we freeze $n-1$ layers to re-train the pre-trained model. We then compare the performance of the attack in the traditional approach to the TF approach using two different scenarios as in the previous experiments.

Results. Table 8 shows the performance of WF attacks using similar but mutually exclusive datasets. The results show that the TF approach performs significantly better compared to the traditional approach when the available number of examples N is small ($N = 1$ or $N = 5$). We observe that as N grows, both approaches perform similarly well with over 93% accuracy. The results suggest that if the attacker has a small dataset to re-train the classifier, TF is the better choice.

[6] http://www.deeplearningessentials.science/transferLearning/

Table 9: *Transfer Learning*: Datasets with different distributions (Accuracy)

Approach	Number of N Example(s)				
	1	5	10	15	20
Traditional	8.6 ± 1.4	31.1 ± 0.9	49.0 ± 0.5	52.5 ± 0.4	56.3 ± 0.9
TF	73.1 ± 1.8	84.5 ± 0.4	86.2 ± 0.4	86.6 ± 0.3	87.0 ± 0.3

On the other hand, the effectiveness of the traditional approach is significantly degraded under the more challenging scenario. Table 9 shows the performance of WF attacks with different distributions of training and testing datasets. By contrast, the performance of the TF attacks is noticeably higher, indicating that the TF approach can better mitigate the negative effects of data mismatch. As we see, with small values of N, TF provides 50% higher accuracy than the traditional approach. Furthermore, even if N grows e.g. to $N = 20$ examples used for training each website, the accuracy of the traditional approach is only 56%, whereas TF reaches 87%. The results suggest that if the attacker knows that the dataset used for pre-training is likely to be dissimilar to the data in the attack phase, then the TF approach is more effective.

Overall, TF seems to be a more reliable option for the attacker. Since the traditional approach works best in this setting when freezing $k = n - 1$ layers, feature extraction learned at the deeper layers appear to be too helpful to throw out. This may mean, however, that transfer learning suffers from not being able to fine-tune more of its layers for the new dataset. TF, on the other hand, is already designed to extract features at all depths of the model, which may help it in this setting.

6.7 Open-World Scenario

In the previous experiments, we explored the performance of the TF attack under the closed-world scenario. However, this scenario is unrealistic, as it assumes that the users will only visit websites within the monitored set. In the following experiment, we evaluate the performance of the TF attack in the more realistic open-world setting. In the open-world, the classifier must learn to distinguish between monitored and unmonitored sites. We use precision and recall metrics to evaluate the performance of the attack.

Experimental Setting. We evaluate the open-world setting under the *standard model* in which we include the unmonitored samples as an additional label during training. We use the AWF100 dataset for the monitored websites and AWF9000 for the unmonitored websites to evaluate WF attacks with similar but mutually exclusive datasets. For the setting when datasets come from different distributions, we use the Wang100 dataset for the monitored websites and AWF9000 for the unmonitored websites, which again represents a three-year gap using different TBB versions.

Results. Figure 6 shows precision-recall curves in the open-world setting for attacks using similar but mutually exclusive datasets, while Table 10 shows the results when the attack is tuned for precision or tuned for recall. The open-world results are fairly effective with moderately high precision and recall. For example, with 10-shot learning, the attacks reach to 0.908 precision and 0.788 recall when tuned for precision, and 0.730 precision and 0.948 recall when

Table 10: Open World: Similar but mutually exclusive datasets

N	Tuned for Precision		Tuned for Recall	
	Precision	Recall	Precision	Recall
5	0.871	0.808	0.804	0.893
10	0.908	0.788	0.730	0.948
15	0.891	0.829	0.692	0.966
20	0.873	0.862	0.706	0.968

Table 11: Open World: Different data distributions

N	Tuned for Precision		Tuned for Recall	
	Precision	Recall	Precision	Recall
5	0.973	0.831	0.950	0.950
10	0.953	0.879	0.922	0.971
15	0.944	0.903	0.908	0.983
20	0.933	0.907	0.905	0.978

tuned for recall. In the case of WF attacks with different distributions, Figure 7 and Table 11 show highly effective attacks. For example, with 10-shot learning, the attacks could reach to 0.953 precision and 0.879 recall when tuned for precision, and 0.922 precision and 0.971 recall when tuned for recall. Overall, TF is effective in both settings.

It may be surprising that the open-world results are better when the distributions are different, which is the opposite of the closed-world results. As mentioned above, however, the performance of WF attacks in the open-world scenario relies on the ability to distinguish between the monitored and unmonitored websites. The monitored sites in the Wang dataset come from a list of websites blocked in China, the UK, and Saudi Arabia, while the unmonitored websites are in the Alexa Top list. We observe that many of these monitored sties have unique content, making them more fingerprintable, and we believe that this led to the inconsistency between the closed-world and open-world scenarios. .

6.8 TF attack in the Larger Open World

In the real world, the number of active websites is about 200 million.[7] To the best of our knowledge, the largest unmonitored set is the 400,000 websites collected by Rimmer et al. (AWF400K) [26].

It is interesting to investigate the performance of the TF attack against different open-world sizes. Using the AWF dataset, we evaluate the precision and recall of the attack against various sizes of the unmonitored set: 50K, 100K, 200K, and 400K. We selectively present the results for 10-shot learning here, and we provide further result details and comparisons in Appendix B.

Table 12 and Figure 8 show the performance of the TF attack with respect to the increased size of the unmonitored set. These results demonstrate that there is a reduction in performance when the attack has to deal with the larger open world. Against the 400K unmonitored websites, the performance of the TF attack drops down to 0.333 precision and 0.639 recall. Against moderate-sized unmonitored sets of sizes 50k and 100k, the TF attack can still provide reasonably effective performance with 0.792 and 0.669 precision, respectively, and ~0.64 recall for $N = 10$.

[7] https://www.internetlivestats.com/total-number-of-websites/, as of August 2019.

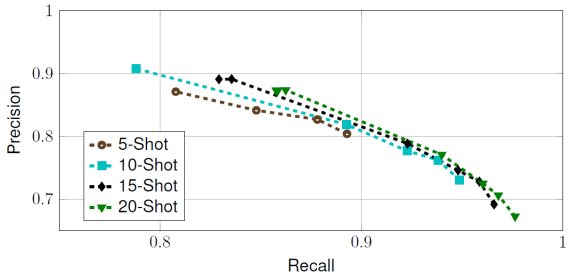


Figure 6: Open-world: Precision and Recall for similar but mutually exclusive datasets

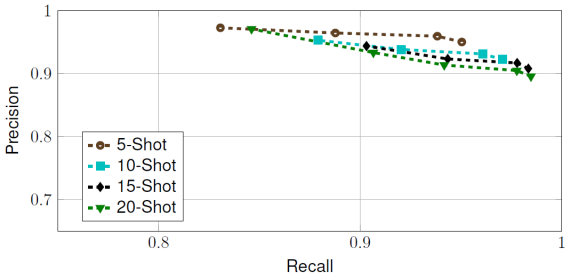


Figure 7: Open-world: Precision and Recall for different data distributions

Table 12: Larger Open World: Growing numbers of unmonitored websites.

Unmonitored Sites	Tuned for Precision		Tuned for Recall	
	Precision	Recall	Precision	Recall
9k	0.908	0.788	0.730	0.948
50k	0.792	0.637	0.591	0.819
100k	0.669	0.639	0.429	0.817
200k	0.484	0.632	0.267	0.794
400k	0.333	0.639	0.297	0.710

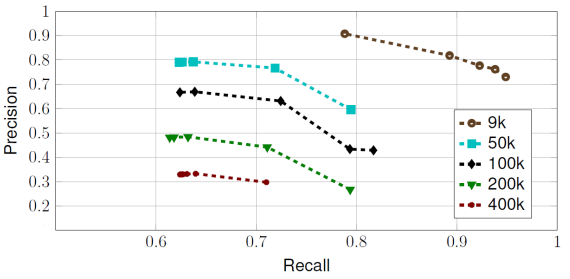


Figure 8: Open-world: Precision and Recall in a large open world with growing numbers of unmonitored sites.

6.9 TF Attack against WTF-PAD

We further evaluate the performance of the TF attack against WTF-PAD, the defense that is the main candidate to be deployed in Tor.

Table 13: The performance of the TF attack against WTF-PAD defense (Accuracy of the attack)

Type of Experiment	N-Example 1	5	10	15	20
Disjointed Websites	20.5 ± 1.6	54.1 ± 0.7	57.8 ± 0.6	60.2 ± 0.4	61.2 ± 0.4
Different Distributions	15.5 ± 1.7	39.8 ± 0.5	47.2 ± 1.1	50.1 ± 0.4	51.7 ± 0.5

Experimental Setting. To train the triplet networks, we simulated defended traces using WTF-PAD. [8] However, we cannot simulate WTF-PAD on the AWF dataset because it contains only packet direction, while WTF-PAD requires timestamps. Therefore, we instead simulated the WTF-PAD traces from the DF95 dataset which contains both packet direction and timing. Furthermore, we use a model trained on the WTF-PAD simulated DF95 dataset and WTF-PAD simulated Wang100 as the N-training and testing datasets, respectively, to evaluate the performance of the TF attack with different data distributions.

Results. Table 13 shows the performance of the TF attack against WTF-PAD defense in two different scenarios. The results show that the accuracy of the attacks in both cases significantly decrease compared to the non-defended dataset in Table 5 and Table 7. As we see, it requires at least 15 examples in the case of WF attacks with similar but mutually exclusive datasets to reach to 60% accuracy. In the case of WF attacks with different data distributions, the performance can only reach to 50% accuracy with 15 examples.

However, if we compare the performance of the TF attack using a small dataset with the previously-proposed WF attacks, the TF attack achieves comparable results. For example, the TF attack (60.2%) performs nearly the same as the CUMUL (60.3%) and AWF (60.8%) attacks. Moreover, it outperforms SDAE (36.9%), *k*-NN (16.0%) and *k*-FP (57.0%).

7 DISCUSSION

Based on what we have observed in our experiments, we now discuss other key benefits of NSL in WF:

WF attacks against interactive websites: Many websites change content frequently, as new articles or other information is posted, and these sites are particularly challenging to fingerprint. NSL allows the attacker to quickly collect mostly up-to-date network traffic examples and immediately use them to train the WF classifier. This may allow the attacker to achieve better performance on these frequently changing sites by using very fresh data that accurately characterizes the current state of the site.

Webpage fingerprinting: Most prior work (with exception of [23]) only performs WF on the homepages of each websites and does not consider all webpages within the website. The TF attack makes the fingerprinting of many webpages from a given website more feasible due to the reduced data requirements.

Threat landscape: The ability to use a pre-trained feature extractor with a few network examples may allow a less powerful adversary to perform WF. Thus, NSL expands the threat landscape, such that the attack is not limited to attackers with significant computing resources.

[8] We synthesized our WTF-PAD dataset using the code provided by the author.

Countermeasures: The results from our WTF-PAD experiment demonstrates that the TF attack is not as effective as the DF attack when traffic is protected. This suggests that light-weight padding mechanisms may be further developed to target NSL attacks.

Limitations: While we were able to investigate the issue of data mismatch during the pre-training and the N-training phases, we were unable to do the same for the N-training and testing phases due to limitations in our datasets. However, we feel this scenario is easily avoided in real world attacks, since the burden of collecting so few fresh samples is low and many WF adversaries (e.g. ISP or compromised router) are in a position to replicate their target's network conditions. In addition, we did not address the problems of session extraction and multi-tab browsing. Recent work [37] has proposed algorithmic stream-splitting and chunk-based classification to address these problems, but attack performance remains inadequate for real-world application. To appropriately address these problems, different techniques will need to be developed.

8 CONCLUSION

In this study, we investigated the use of N-shot learning to improve a WF attack under more realistic and more challenging scenarios than found in most papers on WF attacks. We propose the Triplet Fingerprinting (TF) attack leveraging triplet networks for N-shot learning, which allows an attacker to train on just a few samples per site. We evaluate the TF attack under several challenging scenarios. The results show that the TF attack remains effective with 85% accuracy even in a scenario in which the data used for training and testing are from multiple years apart and collected on different networks. Moreover, we also demonstrate that the TF attack is effective in the open-world setting and can outperform traditional transfer learning. These results demonstrate that an attacker with relatively low computing resources can also perform WF attacks with fairly effective performance.

Reproducibility

The source code of the implementation and a dataset to reproduce our results is publicly available at https://github.com/triplet-fingerprinting/tf.

ACKNOWLEDGMENTS

We appreciate our discussions with Dr. Leon Reznik, Dr. Sumita Mishra and Dr. Peizhao Hu that helped develop this paper. Moreover, we thank the anonymous reviewers for their helpful feedback and Dr. Amir Houmansadr for shepherding our work.

This material is based upon work supported by the National Science Foundation under Award No. 1423163, 1722743 and 1433736.

REFERENCES

[1] 2011. Test Pilot New Tab Study Results. https://blog.mozilla.org/ux/2011/08/test-pilot-new-tab-study-results/. (2011). (accessed: August, 2018).

[2] 2017. Keras. https://keras.io/. (2017).

[3] 2017. Users - Tor metrics. https://metrics.torproject.org/userstats-relay-country.html. (2017).

[4] 2019. New Release: Tor 0.4.0.5. https://blog.torproject.org/new-release-tor-0405. (2019). [Online; accessed May-2019].

[5] K. Abe and S. Goto. 2016. Fingerprinting attack on Tor anonymity using deep learning. In *in the Asia Pacific Advanced Network (APAN)*.

[6] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. 2018. Var-CNN and DynaFlow: Improved Attacks and Defenses for Website Fingerprinting. "https://arxiv.org/pdf/1802.10215.pdf". (2018). (accessed: August, 2018).

[7] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. 2019. Var-CNN: A Data-Efficient Website Fingerprinting Attack Based on Deep Learning. *Proceedings on Privacy Enhancing Technologies* 2019, 4 (2019), 292–310.

[8] F. Chollet. 2017. Xception: Deep Learning with Depthwise Separable Convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1800–1807. https://doi.org/10.1109/CVPR.2017.195

[9] Li Fei-Fei, R. Fergus, and P. Perona. 2006. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28, 4 (April 2006), 594–611. https://doi.org/10.1109/TPAMI.2006.79

[10] Ben Harwood, BG Kumar, Gustavo Carneiro, Ian Reid, Tom Drummond, et al. 2017. Smart mining for deep metric learning. In *Proceedings of the IEEE International Conference on Computer Vision*. 2821–2829.

[11] Jamie Hayes and George Danezis. 2016. k-fingerprinting: A robust scalable website fingerprinting technique. In *USENIX Security Symposium*. USENIX Association, 1–17.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[13] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. 2009. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial Naïve-Bayes classifier. In *ACM Workshop on Cloud Computing Security*. ACM, 31–42.

[14] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. 2014. A critical evaluation of website fingerprinting attacks. In *ACM Conference on Computer and Communications Security (CCS)*. ACM, 263–274.

[15] Marc Juarez, Mohsen Imani, Mike Perry, Claudia Diaz, and Matthew Wright. 2016. Toward an efficient website fingerprinting defense. In *European Symposium on Research in Computer Security (ESORICS)*. Springer, 27–46.

[16] Simonyan Karen and Zisserman Andrew. 2015. Very deep convolutional networks for large-scale image recognition. (2015).

[17] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. 2015. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, Vol. 2.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 1097–1105.

[19] Akshaya Mani, T. Wilson-Brown, Rob Jansen, Aaron Johnson, and Micah Sherr. 2018. Understanding Tor Usage with Privacy-Preserving Measurement. In *Proceedings of the Internet Measurement Conference 2018 (IMC '18)*. ACM, New York, NY, USA, 175–187. https://doi.org/10.1145/3278532.3278549

[20] Alireza Bahramali Milad Nasr and Amir Houmansadr. 2018. DeepCorr: Strong Flow Correlation Attacks on Tor, Using Deep Learning. In *ACM Conference on Computer and Communications Security (CCS)*. ACM, 1962–1976. https://doi.org/10.1145/3243734.3243824

[21] Se Eun Oh, Saikrishna Sunkam, and Nicholas Hopper. 2018. *p*-FP: Extraction, Classification, and Prediction of Website Fingerprints with Deep Learning. "https://arxiv.org/abs/1711.03656.pdf". (2018). (accessed: August, 2018).

[22] Se Eun Oh, Saikrishna Sunkam, and Nicholas Hopper. 2019. p-FP: Extraction, Classification, and Prediction of Website Fingerprints with Deep Learning. *Proceedings on Privacy Enhancing Technologies* 2019, 3 (2019), 191–209.

[23] Andriy Panchenko, Fabian Lanze, Andreas Zinnen, Martin Henze, Jan Pennekamp, Klaus Wehrle, and Thomas Engel. 2016. Website fingerprinting at Internet scale. In *Network & Distributed System Security Symposium (NDSS)*. IEEE Computer Society, 1–15.

[24] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. 2015. Deep face recognition.. In *BMVC*, Vol. 1. 6.

[25] Mike Perry. 2015. Padding Negotiation. Tor Protocol Specification Proposal. https://gitweb.torproject.org/torspec.git/tree/proposals/254-padding-negotiation.txt. (2015). (accessed: October 1, 2017).

[26] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. 2018. Automated Website Fingerprinting through Deep Learning. In *Proceedings of the 25nd Network and Distributed System Security Symposium (NDSS 2018)*. Internet Society.

[27] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. FaceNet: A Unified Embedding for Face Recognition and Clustering. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[28] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. 2017. Beauty and the Burst: Remote identification of encrypted video streams. In *USENIX Security Symposium*. USENIX Association, 1357–1374.

[29] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. 2018. Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning. *The 25th ACM SIGSAC Conference on Computer and Communications Security (CCS '18)* (2018).

[30] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. 2015. RAPTOR: Routing Attacks on Privacy in Tor.. In *USENIX Security Symposium*. 271–286.

[31] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. (June 2015).

[32] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. 2014. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1701–1708.

[33] C. v. d. Weth and M. Hauswirth. 2013. DOBBS: Towards a Comprehensive Dataset to Study the Browsing Behavior of Online Users. In *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, Vol. 1. 51–56. https://doi.org/10.1109/WI-IAT.2013.8

[34] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. 2016. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*. 3630–3638.

[35] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective attacks and provable defenses for website fingerprinting. In *USENIX Security Symposium*. USENIX Association, 143–157.

[36] Tao Wang and Ian Goldberg. 2016. On realistically attacking Tor with website fingerprinting. In *Proceedings on Privacy Enhancing Technologies (PoPETs)*. De Gruyter Open, 21–36.

[37] Yixiao Xu, Tao Wang, Qi Li, Qingyuan Gong, Yang Chen, and Yong Jiang. 2018. A Multi-tab Website Fingerprinting Attack. In *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC '18)*. ACM, New York, NY, USA, 327–341. https://doi.org/10.1145/3274694.3274697

[38] Junhua Yan and Jasleen Kaur. 2018. Feature Selection for Website Fingerprinting. In *Proceedings on Privacy Enhancing Technologies (PETS)*.

[39] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How Transferable Are Features in Deep Neural Networks?. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'14)*. MIT Press, Cambridge, MA, USA, 3320–3328. http://dl.acm.org/citation.cfm?id=2969033.2969197

[40] Ye Zhu, Xinwen Fu, Bryan Graham, Riccardo Bettati, and Wei Zhao. 2010. Correlation-based traffic analysis attacks on anonymity networks. *IEEE Transactions on Parallel and Distributed Systems* 21, 7 (2010), 954–967.

A DATASET'S DISTRIBUTION ANALYSIS

In Section 6.5, we study the performance of the triplet fingerprinting attack under more challenging scenarios. One of the experiments that we aim to evaluate is the performance of the attack against the adverse effect causing from a data mismatch issue. The issue happens when the distributions of training and testing datasets are significantly different. We use two different datasets including 1) AWF dataset collected in 2016 using Tor browser version 6 as the training data, and 2) Wang dataset collected in 2013 using Tor browser version 3 as the testing data.

Even if the three-year gap of different periods and three different versions of Tor browsers between AWF and Wang datasets can presumably ensure the significant difference in term of their distributions, we provide further investigation to evaluate how significantly different between these datasets are. We provide evaluations based on two metrics including the basic network traffic statistic and the similarity measurement.

A.1 Basic Network Traffic Statistic

Common Websites in Wang and AWF datasets: We analyze the network traffic statistic on websites that are commonly listed

Table 14: The comparison between the average number of incoming and outgoing packets for common websites in the Wang and the AWF datasets

Websites	Number of Packets			
	Wang Dataset		AWF Dataset	
	Outgoing	Incoming	Outgoing	Incoming
nicovideo.jp	307	4016	255	2584
youtube.com	153	1896	291	3094
rakuten.co.jp	48	478	382	4617
dropbox.com	142	2457	436	4286
xvideos.com	165	2983	139	1536
excite.co.jp	356	3346	414	2504
nikkeibp.co.jp	413	3250	423	4572
archive.org	26	87	118	1421
twitter.com	59	717	163	2214
t.co	17	48	22	47
scribd.com	376	3896	354	2791
facebook.com	75	866	101	795
appspot.com	63	314	74	255
vimeo.com	203	3318	425	4574
soundcloud.com	176	2043	673	3115
drtuber.com	289	3322	329	4119
imgur.com	160	568	366	4340
fc2.com	125	1255	131	1042
wordpress.com	103	988	347	1367
imdb.com	268	1886	427	4550
xhamster.com	211	2180	173	1723
xing.com	80	740	208	878
extratorrent.cc	237	2028	259	2836

Table 15: The comparison between the average number of incoming and outgoing bursts for common websites in the Wang and the AWF datasets

Websites	Number of Bursts			
	Wang Dataset		AWF Dataset	
	Outgoing	Incoming	Outgoing	Incoming
nicovideo.jp	116	116	131	131
youtube.com	69	68	157	157
rakuten.co.jp	21	21	200	200
dropbox.com	68	67	233	233
xvideos.com	72	72	70	70
excite.co.jp	131	131	208	207
nikkeibp.co.jp	120	120	247	247
archive.org	8	8	62	62
twitter.com	32	31	103	102
t.co	9	9	11	11
scribd.com	147	146	168	168
facebook.com	32	32	53	53
appspot.com	32	32	40	40
vimeo.com	92	92	231	230
soundcloud.com	72	72	293	292
drtuber.com	88	88	185	184
imgur.com	59	59	197	197
fc2.com	55	55	70	70
wordpress.com	46	45	165	164
imdb.com	81	80	225	225
xhamster.com	59	59	86	86
xing.com	38	38	99	98
extratorrent.cc	91	91	144	144

in the Wang and the AWF datasets. We first evaluate the number of incoming and outgoing packets which directly represents the change in each website's contents in term of the size of the website over periods of time. Table 14 demonstrates the average numbers of outgoing and incoming packets for each website. The results show that there are significant changes in term of the number of outgoing and incoming packets in most of the websites.

Moreover, we take a closer look on the burst-level packets statistic by measuring the number of incoming and outgoing bursts. It is important to note that the number of bursts is commonly used by WF classifiers to measure the distinctive patterns of the traffic and the network's activities such as the total number of objects from which the user needs to fetch (number of requests and responds for each given website). Table 15 demonstrates the significant changes in term of the number of incoming and outgoing bursts for common websites in the Wang and the AWF datasets. Therefore, the significant changes of the basic network statistic and bursts for the common websites in the Wang and the AWF datasets reveals the likelihood of mismatched data.

Aggregation Analysis of Wang and AWF datasets: We further investigate aggregation analysis of the basic network traffic statistic in all websites from both datasets. Instead of focusing on the common websites, we measure the traffic statistic for all 100 websites of each dataset. The results of the aggregation analysis provide the statistical analysis for the whole datasets used for training and testing the classifier. Table 16 and Figure 9- 10 show the significant difference on both the number of network's packets and the number of the traffic's bursts. The results support the evidence that there is a mismatch of the AWF dataset used for training and the Wang dataset used for testing the classifier.

A.2 Similarity Measurement

We extensively analyze the difference of the two datasets by using a similarity measurement. The similarity measurement is used to evaluate how similar of two network traffic's vectors in the latent space. We apply the Cosine distance to calculate the distance between a pair of network traffic; the smaller distance represents the higher similarity of the given pair of network traffic.

In the set of common websites in both Wang and AWF datasets, we first measure the *intrasimilarity* in which the Cosine distance is calculated for all possible pairs of the same website. Moreover, we additionally measure the *intersimilarity* in which we calculate the Cosine distance for all possible pairs of network traffic examples from website A in the Wang dataset and network traffic examples from website A in the AWF dataset. The illustration of how the

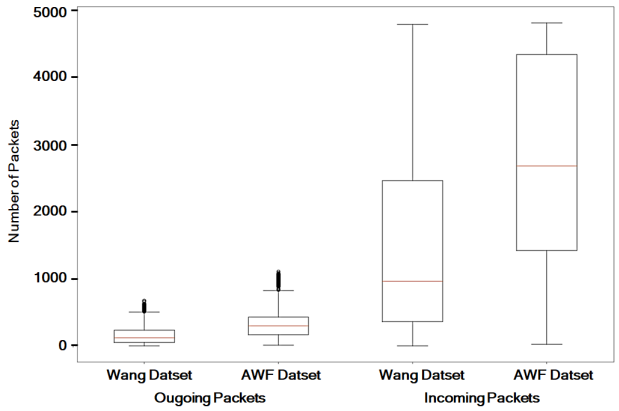


Figure 9: The box plot of the basic statistical analysis in term of the number of packets for all websites in the Wang and the AWF datasets

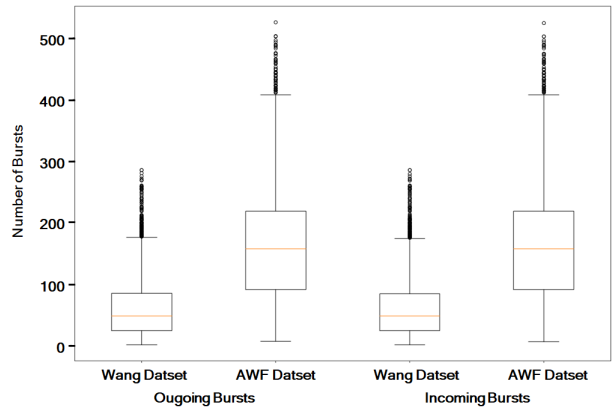


Figure 10: The box plot of the basic statistical analysis in term of the number of bursts for all websites in the Wang and the AWF datasets

pairs of network traffic examples are used to measure the similarity is demonstrated in Figure 11.

Table 17 demonstrates the intersimilarity and the intrasimilarity resulted from the average of similarity from all pairs of network traffic examples. The results show that there are significant reductions of the intersimilarity in many common websites compared to the intrasimilarity of them. Thus, it can be inferred that the similarity for the same website in different datasets has significantly decreased in the latent space supporting the evidence that the common websites in Wang and AWF dataset are likely to be mismatched.

B LARGER OPEN-WORLD SCENARIO

We comprehensively evaluate the performance of the TF attack against the larger open world to measure the realistic capability of the attack when the size of unmonitored websites (UMW) increases. This reflects a realistic challenge in the real world since the larger

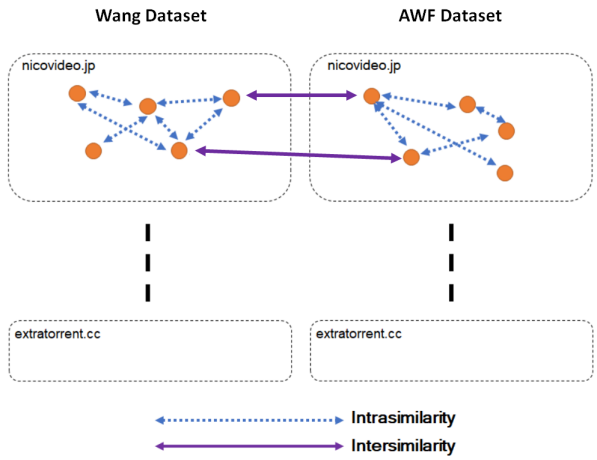


Figure 11: Similarity measurement by using the intrasimilarity and the intersimilarity based on the Cosine distance metric

Table 16: The aggregation analysis of the basic network statistical analysis for all websites in Wang and AWF datasets (the average of the number of incoming and outgoing packets vs. the average of the number of incoming and outgoing bursts)

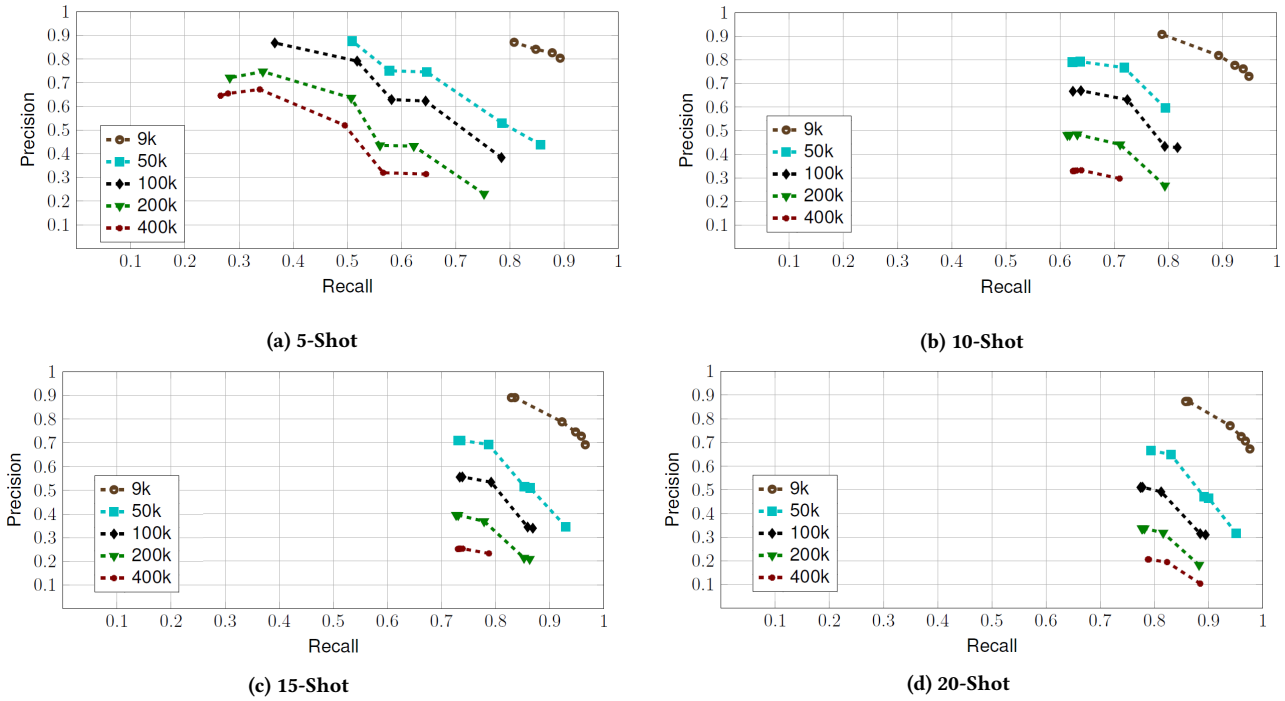
Dataset	Number of Packets		Number of Bursts	
	Outgoing	Incoming	Outgoing	Incoming
Wang	156	1532	59	59
AWF	315	2735	160	160

the size of the UMW is, the higher the chance of coalition of the similar websites, resulting the increased chance of misclassification. We experiment the open-world scenario with growing sizes of UMW including 9k, 50k, 100k, 200k, and 400k by using Rimmer et al. dataset. In each of N-Shot learning, we demonstrate the precision and recall of the TF attack against different size of UMW. We compare them side by side as shown in Figure 12. The results demonstrate the impact of the larger open world in two folds. First, we observe the reduction of performance of the TF attack with respect to the increased size of UMW for each of N-Shot learning. Second, the precision and recall have improved when the size of the number of N examples (N-Shot) increases. Overall, the attack still remains fairly effective till the size of unmonitored websites reaches to 100k with almost ~0.6–0.7 precision and ~0.6–0.9 recall.

C TOP-*N* AND *K*-WAY ACCURACY

In this part, we further investigate the performance of the TF attack under the closed-world scenario by using Top-*n* and *k*-Way accuracy.

Top-n: The top-*n* accuracy is the percentage of correct predictions in which we consider not only the highest probability, but also the top-*n* probability values. In WF attacks, using top-*n* prediction



(a) 5-Shot (b) 10-Shot

(c) 15-Shot (d) 20-Shot

Figure 12: Open World: Each sub-figure shows the Precision-Recall of the TF attack using different N-Shot learning against different sizes of unmonitored websites.

helps the attacker to scope down n possible websites that the user may visit.

k-Way: The k-Way accuracy evaluates classification performance at particular sizes of monitor set—e.g. 100 monitored sites is 100-way accuracy. It is interesting to investigate how the attack accuracy may improve if the attacker targets a limited set of monitored websites.

Experimental Setting: In the Top-n and k-Way accuracy, we set n = [1, 2, 5] and k = [100, 75, 50, 25] to evaluate the performance of the attacks for all possible combinations of n and k with different number of N example(s). To effectively demonstrate the impact of Top-n accuracy, we fixed the size of k-Way to be 100. Likewise, we show the impact of k-Way accuracy by fixing the Top-n where n=1.

Result:

- **WF with similar but mutually exclusive datasets**: As we see from Figure 13, we observe that the consistent improvement in Top-2 and Top-5 predictions where N >= 5 and the accuracy can reach up to 95% accuracy in Top-2 prediction with 5-Shot learning. For k-Way results, the smaller size of problem demonstrates the improvement of accuracy e.g. 50-Way with 5-Shot learning, the accuracy slightly improves from 100-Way with ~2%. Moreover, we observe that 1-Shot learning gains larger increase with ~4% accuracy compared to others.
- **WF with different data distributions**: According to Figure 14, we observe that the attacks have larger increase compared to the WF attacks regardless of website's label especially with N >= 5. In terms of Top-n prediction, from Top-1 to Top-2 and from Top-1 to Top-5, the accuracy consistently improves with ~5% and ~10% respectively. For k-Way prediction, the accuracy of the attack gradually improves as k gets smaller except for 1-Shot learning that gains significant increase with ~5% accuracy when k is reduced from k = 100 to k = 50.

Overall, the results show the consistent increase of the accuracy with the larger Top-n and smaller k-Way. The improved accuracy form Top-1 to Top-2 can be interpreted that the triplet model cannot perform well enough to move the embedded vector to be closest to the corresponding website in the embedding space. However, it is closed enough to be in the second closest neighbor. This suggests that the improvement of the subnetwork or the larger number of websites used for training model can possibly be the key factors to substantially improve quality of the model and ultimately increase the performance of the attacks. The results also reveal that if the attacker is interested in detecting a smaller set of websites, the performance of the attack can be improved.

D TF ATTACK'S HYPERPARAMETER TUNING PROCESS

In this part, we provide intuitive explanation and the preliminary results used for each parameter selection in addition to what is mentioned in Section 5.1. For each selection, we vary the range of parameters to run the model. We only use the subset of training and testing data due to the fact that using the full size of data requires large amount of time to complete each evaluation. We then

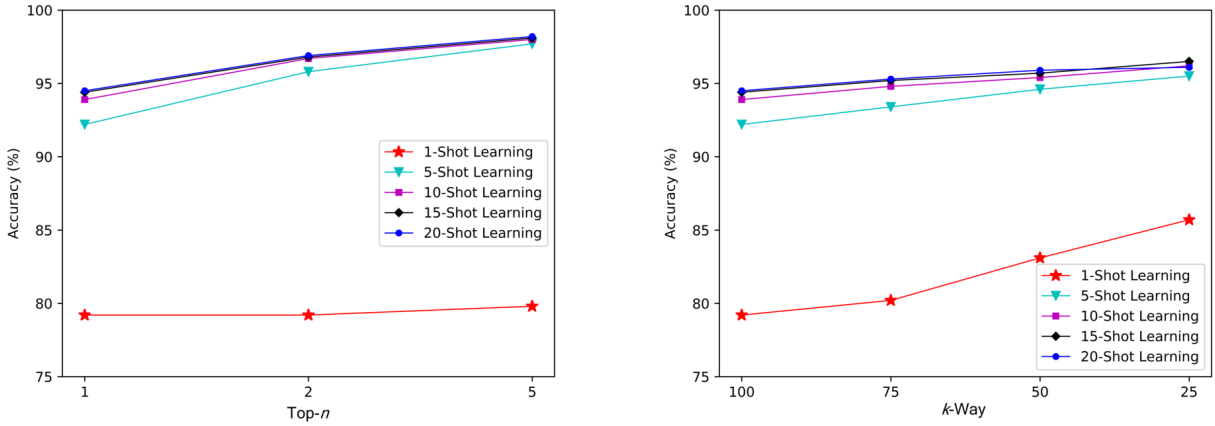


Figure 13: The Top-*n* (Left) and the *k*-Way (Right) accuracy of WF attacks with similar but mutually exclusive datasets

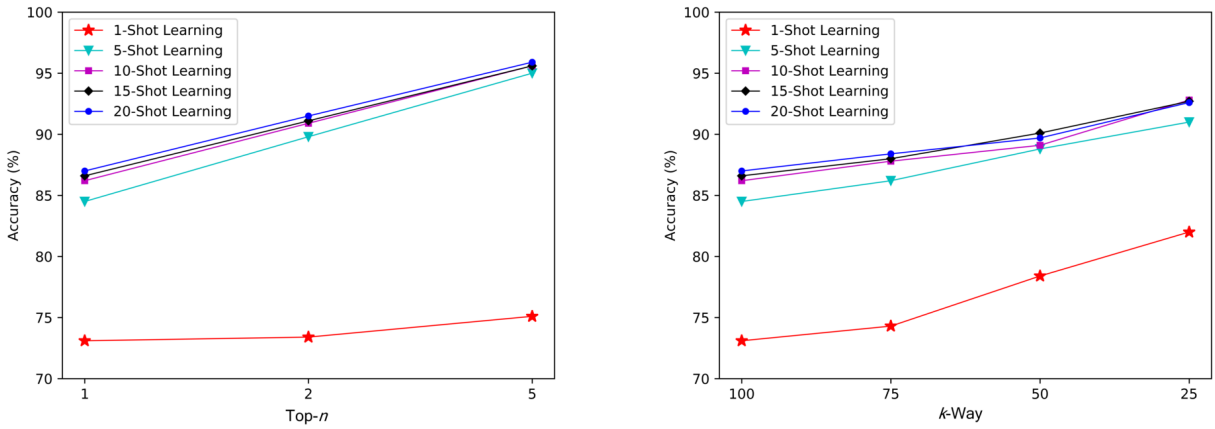


Figure 14: The Top-*n* (Left) and the *k*-Way (Right) accuracy of WF attacks with different data distributions.

compare the result given by each setting and finally selected the most effective one.

- **Based Model**: We test with previously-proposed state-of-the-art DL models used in image recognition literature including GoogleNet [31], RestNet [12] and Xception [8] and compared with DF [29] model; the state-of-the-art model dedicatedly designed for the WF task. Our preliminary results demonstrate that the DF model can perform slightly better with significantly less training time required. It turns out that using the DF as the base model takes approximately 400s on each epoch of training. On the other hand, the very deep learning models such as GoogleNet, RestNet and Xception require over 600s on each epoch of training. Thus, we choose the DF model as the subnetwork in the triplet networks.
- **Distance Metrics**: We compare two distance metrics generally used in the N-Shot learning applications including Cosine distance and Euclidean distance. Our preliminary results shows that using the Cosine distance provides higher performance than using the Euclidean one with ~2–3% better accuracy.
- **Mining Strategy**: We evaluate three different mining strategies used to generate the triplets input including *Random*, *Hard-Negative* and *Semi-Hard-Negative*. The results shows that the *Semi-Hard-Negative* perform better than the rest in term of the accuracy of the attack e.g. ~7% and ~40% when compared with *Random* and *Hard-Negative* minings respectively.
- **Margin**: The margin's value defines a radius around the embedded vector and helps for the learning process. The value is specific for each type of data's input, thus, we have to find the best m value that is suitable for network traffic and provide effective performance to the model. We find that margin's value $m = 0.1$ can provide better results than others with ~0.3–2.9% higher accuracy.
- **Optimizer**: The optimizer is a mathematical model used to measure and update the weights' learning with respect to the

Table 17: The comparison between the intrasimilarity and the intersimilarity for common websites in the Wang and the AWF datasets

Websites	Intrasimilarity		Intersimilarity
	Wang	AWF	Wang - AWF
nicovideo.jp	0.7	0.7	0.56
youtube.com	0.64	0.63	0.55
rakuten.co.jp	0.74	0.73	0.19
dropbox.com	0.8	0.66	0.57
xvideos.com	0.66	0.68	0.53
excite.co.jp	0.62	0.49	0.51
nikkeibp.co.jp	0.6	0.7	0.55
archive.org	0.52	0.72	0.11
twitter.com	0.76	0.57	0.49
t.co	0.33	0.32	0.2
scribd.com	0.66	0.63	0.56
facebook.com	0.69	0.64	0.62
appspot.com	0.52	0.4	0.35
vimeo.com	0.71	0.69	0.61
soundcloud.com	0.72	0.5	0.56
drtuber.com	0.71	0.72	0.65
imgur.com	0.38	0.72	0.19
fc2.com	0.71	0.62	0.6
wordpress.com	0.66	0.42	0.45
imdb.com	0.56	0.69	0.39
xhamster.com	0.74	0.69	0.6
xing.com	0.71	0.43	0.48
extratorrent.cc	0.55	0.65	0.54

loss model. We evaluate different types of optimizer used during the training process including *SGD*, *ADAM*, *Adamax* and *RMProp*. SGD performs best with slightly higher performance compared to others with ∼0.3–1.7% better accuracy. Thus, we choose SGD as optimizer used during the model's training process.

- **Batch Size**: Batch size is the number of inputs that are sampled to be fed to the network during the training process. We find that $batch\ size = 64$ and $batch\ size = 128$ can provide ∼1.5% better accuracy than others. Thus, we choose to use them as the final choices.
- **Embedded Vector's Size**: The embedded vectors size is the size of the last dense layer in each subnetwork. It contains the vectors of the output after feeding the input through the learned model to extract features. We find that there is no noticeable difference between our candidates. Thus, we simply select the $embedded\ vectors\ size = 64$ as the final selection because it requires less number of training parameters in the network helping reduce the time for training the model.