Towards More Effective Traffic Analysis in the Tor Network.

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Se Eun Oh

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Nicholas Hopper

February, 2021

Acknowledgements

Above all, I would like to extend my deepest gratitude to my advisor, Nick Hopper, for his valuable advising and boundless encouragement. His eternal support always made me thrilled to conduct the anonymity research throughout the graduate program. I have learned a lot from his insightful suggestions as well as from his profound knowledge.

I also wish to thank other committee members, Stephen McCamant, Kangjie Lu, Andrew Odlyzko, and Matthew Wright, for their time and support. In particular, I thank Matthew Wright who allowed me to visit his lab as a visiting scholar. It was not only beneficial, but also enjoyable to communicate with other cyber security researchers.

I'm grateful to all my research collaborators, Shuai Li, Erik Lindeman, Nate Mathews, Mohammad Saidur Rahman, Saikrishna Sunkam, Danyang Wang, and Taiji Yang, for their helpful contributions to all my research projects. I also thank my lab mates for helpful research and personal discussions, John Geddes, James Holland, Shuai Li, Saugata Paul, Mike Schliep, Max Schuchard, and Jaskaran Veer Singh.

Last but not least, I'm deeply indebted to my family. I appreciate my parents, Chang Seok Oh and In Gyu Kwack, for their never-ending support and love. I'm extremely grateful to my wonderful husband and another invaluable collaborator, Ki Bum Noh, for his endless support to take care of our kids and patience towards my busy timeline. It would have been impossible to maintain a work-life balance without him. Special thanks to my amazing daughter, Grace Noh, who has been patient with me during my entire graduate life and helped a lot to take care of my lovely little son, Daniel Noh.

Dedication

To my lord, Jesus Christ for his love

Abstract

Tor is perhaps the most well-known anonymous network, used by millions of daily users to hide their sensitive internet activities from servers, ISPs, and potentially, nation-state adversaries. Tor provides low-latency anonymity by routing traffic through a series of relays using layered encryption to prevent any single entity from learning the source and destination of a connection through the content alone.

Nevertheless, in low-latency anonymity networks, the timing and volume of traffic sent between the network and end systems (clients and servers) can be used for traffic analysis. For example, recent work applying traffic analysis to Tor has focused on *website fingerprinting*, which can allow an attacker to identify which website a client has downloaded based on the traffic between the client and the entry relay.

Along with website fingerprinting, *end-to-end flow correlation* attacks have been recognized as the core traffic analysis in Tor. This attack assumes that an adversary observes traffic flows entering the network (Tor flow) and leaving the network (exit flow) and attempts to correlate these flows by pairing each user with a likely destination.

The research in this thesis explores the extent to which the traffic analysis technique can be applied to more sophisticated fingerprinting scenarios using state-of-the-art machine-learning algorithms and deep learning techniques. The thesis breaks down four research problems. First, the applicability of machine-learning-based website fingerprinting is examined to a search query keyword fingerprinting and improve the applicability by discovering new features. Second, a variety of fingerprinting applications are introduced using deep-learning-based website fingerprinting. Third, the work presents *data-limited fingerprinting* by leveraging a generative deep-learning technique called a generative adversarial network that can be optimized in scenarios with limited amounts of training data. Lastly, a novel deep-learning architecture and training strategy are proposed to extract features of highly correlated Tor and exit flow pairs, which will reduce the number of false positives between pairs of flows.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Online activities that include details of the user's life may be targeted for censorship or surveillance. As a defense, users rely on encrypted traffic and anonymous networks to conceal their connections. However, no security protections hide the size and timing of traffic between a client and server, As a result, website fingerprints are created allowing machine learning (ML) algorithms to identify which websites the users visited. Tor is one of the most well-known anonymous networks used by millions of daily users. It relays traffic through three proxies and each proxy is only aware of the subsequent destination and prior source to hide the user's identity and destinations from network surveillance conducting traffic analysis. However, researchers have shown that Tor does not guarantee user anonymity because it allows network adversaries to identify which website the user visited based on the size and timing of traffic between the client and server. This process is called website fingerprinting (WF). Adversaries are also able to observe traffic flows entering and leaving the network. They then attempt to correlate these flows by pairing each user with a likely destination. This thesis explores the extent to which the traffic analysis technique can be extended to achieve more advanced fingerprinting goals. The research of the thesis was inspired by four research problems which will be detailed in Section 1.1 and aims to solve these problems.

1.1 Research Problems

- The queries a user makes to search engines necessarily contain a great deal of private and personal information about the user and popular search engines such as Bing and Google are in a position to collect sensitive details about users. The first part of the thesis studies the extent to which a WF can be extended to different attack scenario we call keyword fingerprinting (KF) differentiating between search query traffic from a website. This attack eventually identifies search keywords the users typed, which further enables the adversaries to infer many details of users' private information. This work investigates the feasibility of KF across a variety of experimental settings.

- Since 2017, WF researchers [4, 2] have become more interested in the applicability of deep-learning models that apply to WF classifiers due to their powerful classification capability. The second part of the thesis investigates this problem and further presents three deep-learning applications for WF research by utilizing both generative and discriminative deep neural networks (DNNs). Our applications will include automated feature extraction for traditional machine-learning (ML) algorithms, website fingerprint classification, and website fingerprintability analysis.

- Although the use of DNNs have led to more accurate classification results than traditional ML-based WF, the effectiveness of DNN classifiers can be only guaranteed with an enormous training set that has to be frequently updated to maintain good quality classifiers. Thus, the third part of the thesis focuses on developing classifiers supporting low-data training through the capability of generative adversarial networks (GANs), which are one of the most impactful generative DNNs in the last decade.

- By monitoring both ends of the Tor network, we can correlate these flows using traffic analysis to deanonymize the user and destination, called end-to-end flow correlation. Researchers [5, 6] have investigated this problem by comparing all N^2 incoming and outgoing flows to identify N correlated pairs. This pairwise nature makes the attack even more unrealistic due to a low *base rate* (i.e., $\frac{1}{N}$) probability

that both end flows are actually correlated. The fourth part of the thesis proposes a novel flow correlation framework using two DNNs jointly trained on optimal N triplets, resulting in much lower complexity while achieving superior performance against state-of-the-art attacks.

1.2 Contributions and Outline

The thesis consists of four research problems and corresponding key contributions are summarized. The organization of the thesis is as follows.

Search Query Traffic Analysis (Chapter 3)
This chapter first empirically shows that traditional WF features [7, 8, 9] have not worked for KF. We also discuss the task-specific, new feature set to make a KF effective, leading to a svmResp classifier. After evaluating svmResp in various experimental settings, we discuss its effectiveness across settings. Lastly, the chapter shows why certain keywords are more fingerprintable than others based on the fingerprintability analysis of keywords.

Traffic Analysis with Deep Learning (Chapter 4)
This chapter introduces three applications using traffic analysis with deep learning. First, by leveraging the generative ability of an autoencoder (AE), the first part of the chapter shows that the feature engineering used in traditional machine-learning algorithms can be automated. Incorporating AE techniques have made all state-of-the-art WF based on machine learning algorithms more effective and efficient. In the second part of the chapter, classifiers based on convolutional neural network (CNN) models are evaluated using various fingerprinting scenarios including WF over a Tor network, KF over a Tor network, WF over TLS-encrypted traffic, and WF over defended Tor traffic. These classifiers perform effectively across various experimental scenarios. The third part of the chapter presents more influential website design-level features that impact the fingerprintability of websites. These features can help guide website developers to design better websites that are robust against WF attacks.

Data-Limited Traffic Analysis (Chapter 5)

The primary contribution of this chapter is to explore GANs in a semi-supervised setting and devise data-limited WFs by leveraging a vast amount of unlabeled data and a few labeled samples. The chapter introduces GAN for data-limited fingerprinting (GANDaLF). This technique requires only a few samples of labeled data to train GAN in a semi-supervised fashion. Since we can use any public WF datasets for unlabeled data, we have to gather only a very small training dataset to train the classifier. This approach greatly reduces the efforts researchers need to collect the WF dataset. The chapter shows the effectiveness of GANDaLF in two WF scenarios: to fingerprint the front pages (i.e., index webpages) of websites, and to fingerprint the subpages (i.e., non-index webpages) of websites. The results are more extensive compared to related data-limited WF studies [10, 11].

More Efficient Correlated Flow Traffic Analysis (Chapter 6)

This chapter introduces a new end-to-end flow correlation attack on Tor, which is more scalable and practically effective than state-of-the-art attacks. The chapter first details the primary challenge in applying prior work to large-scale traffic analysis (i.e., low Bayesian detection rates (BDRs) due to the pairwise nature of flow correlation attacks). It then proposes DeepCoFFEA, which stands for Deep Correlated Flow Feature Extraction and Amplification, to reduce false positives (FPs), leading to higher BDRs. By extending the triplet network (or feature embedding network) to be a suitable feature extractor for amplified flow correlation, DeepCoFFEA is based on a pair of FENs, which are jointly trained using the triplet loss function. By evaluating DeepCoFFEA in various experimental settings, the chapter demonstrates that this new architecture and attack paradigm significantly improves state-of-the-art flow correlation attacks at the cost of acceptable time complexity.

Chapter 2

Background

2.1 Tor

Tor is a popular low-latency anonymous network system involving thousands of relays, with eight million daily users [12]. A Tor client creates a *circuit* of three Tor relays randomly selected to communicate with the destination such as websites. This set of relays consists of a guard node that the Tor client connects to, an exit node which connects to the destination, and a middle node that is located between the guard node and exit node. In this way, Tor provides anonymity protection for users since each connection only knows its predecessor and successor but no other entities in the circuit. Furthermore, Tor encrypts the traffic using layered encryption to prevent network-level attackers such as ISPs from decoding the traffic.

Tor is designed to be robust against traffic analysis because all traffic between the client and the guard node is communicated through a single TLS connection and Tor flows down all connections in 512-byte *cells* which is a unit of communication in Tor. However, while Tor provides anonymity against basic traffic inspection, it has been shown that more sophisticated traffic analysis based on machine-learning algorithms and DNNs can be used to recover some information about Tor traffic. This thesis focuses on two types of traffic analysis that predate Tor anonymity, website fingerprinting (Section 2.2) and end-to-end flow correlation attacks (Section 2.3).

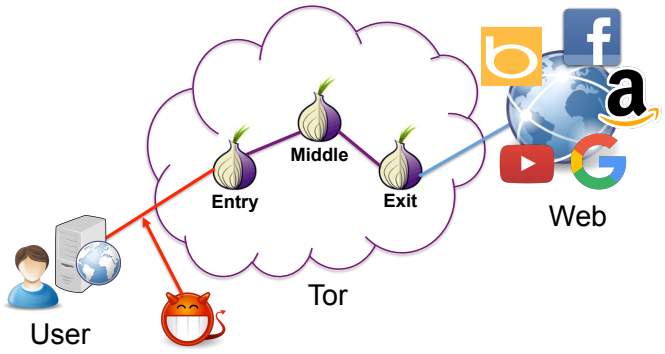


Figure 2.1: Threat model of website fingerprinting.

2.2 Website Fingerprinting

Many security researchers have demonstrated that Tor is somewhat vulnerable to many network-level traffic analysis attacks. It is particularly vulnerable to website fingerprinting attacks that monitor and analyze network traffic between a Tor entry guard and the client to identify the websites visited by the user. To perform this attack, as shown in Figure 2.1 the adversary first makes a series of connections to sites of interest and saves the traffic patterns to create a labeled dataset consisting of pairs: `(traffic_pattern, website)` The adversary then uses this dataset to train a machine-learning classifier to recognize these websites based on traffic patterns observed from the victim's online activity. This type of attack can be successful because the observable traffic patterns across multiple visits to the same webpage are relatively consistent and often fairly distinct from the patterns seen when visiting other sites. Because these attacks depend solely on traffic metadata (rather than the traffic content), WF attacks are successful even when the traffic is encrypted or under the protection of privacy enhancing technologies such as VPNs or Tor.

An important consideration in the study of WF is whether an evaluation uses a *closed-world* setting or an *open-world setting*. In a closed-world setting, which is typically used to make baseline comparisons between models, the victim is assumed to be visiting one of a fixed set of sites that the attacker is interested in and can train on, known as the *monitored set*. In contrast, an experiment in the open-world setting also uses a large set of sites outside of the monitored set, known as the *unmonitored set*, and

allows the victim to visit a site in either set. In practice, a user could be visiting any site on the web, so an open-world evaluation model is a more realistic scenario. It should be noted, however, that open-world evaluations cannot test every possible unmonitored site on the Web – attacks tend to degrade in accuracy as larger unmonitored sets are used in an evaluation. Additionally, neither scenario considers the rate at which users actually visit different sites while using Tor (the *base rate* of each site). Attacks will be more accurate in practice when including popular sites in the monitored set than when looking for less popular sites.

2.2.1 WF Attacks

The first WF attack on Tor was introduced by Herrmann et al. [13] using a Naive Bayes Classifier based on the packet length frequency. Panchenko et al. [14] further improved WF performance by using a support vector machine (SVM) and investigating various feature sets. Other researchers [9, 7, 8] have since explored different classifiers and new feature sets to develop more effective WF models. Wang et al. [7] adapted a k-Nearest Neighbor (k-NN) classifier for classification by applying different weights to different features to accommodate the large feature set in a more effective manner. By working with a reference set for classification, k-NN was also able to model the multi-modal nature of some websites, enabling the attack to achieve closed-world accuracy of 95% on a set of 100 possible websites.

Panchenko et al. [8] significantly improved the performance of the SVM classifier with a new feature set, called CUMUL, based on the cumulative sizes of TCP packets, lengths of TLS records, and number of Tor cells. Hayes and Danezis [9] proposed the k-fingerprinting (k-FP) attack, which leverages a Random-Forest classifier and a statistics-based feature set to achieve high performance. In their open-world evaluation, they computed the hamming distances between RF leaves and used these in a k-NN classifier. This allowed them to tune their attack towards either high precision or high recall by varying the value of k.

2.2.2 WF Defenses

To defend against these ever-more powerful attacks, researchers have devised methods to confuse the attacker's classifiers while aiming to keep bandwidth and latency overheads reasonable. BuFLO [15] adds dummy packets to fill in timing gaps and further extends the transmission to send packets of fixed length at fixed intervals. Tamaraw [16] improves BuFLO to be a more efficient and effective defense by using different padding intervals for incoming and outgoing packet directions and sending outgoing packets at a lower rate, which reduces the overhead in the common case of infrequent outgoing traffic. WTF-PAD [17] is a lighter-weight defense that focuses on hiding large gaps between traffic bursts by adding fake bursts. WTF-PAD significantly decreases bandwidth overhead and latency compared to BuFLO and Tamaraw while yielding good performance against the k-NN attack. Walkie-Talkie [18] is an efficient WF defense technique based on half-duplex communication, which makes many packet sequences the same, and burst molding, which adds fake cells to mold burst sequences into identical supersequences.

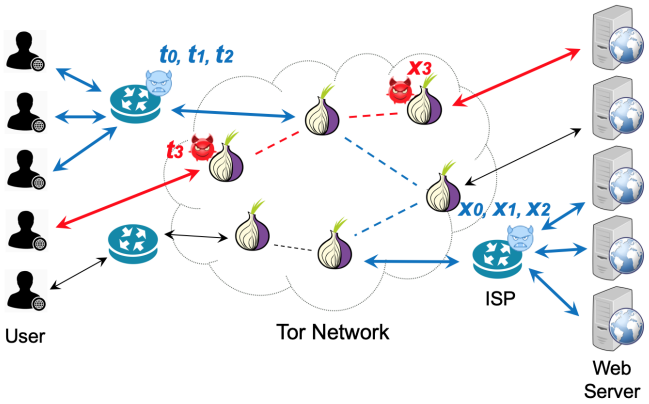


Figure 2.2: Threat model of the end-to-end correlation attack: two types of adversaries: one controls ISPs (blue) and the other runs their own relays (red).

2.3 End-to-End Flow Correlation

Perhaps the most fundamental traffic analysis attack on a low-latency anonymity system is the end-to-end flow correlation attack: an adversary observes traffic flows entering the network and leaving the network and attempts to correlate these flows, pairing each

user with a likely destination (Figure 2.2). Such attacks were known and discussed in the context of system designs that predate Tor such as the Onion Routing network [19] and the Freedom network [20].

End-to-end flow correlation attacks are mentioned in some of the earliest work on low-latency anonymous communications. They are typically referred to as Last/First attacks or Packet Counting attacks [19, 20, 21]. Since designs like the Freedom Network and Tor introduce a basic amount of padding that defeats simple packet counting, later works on passive end-to-end attacks used statistical measures of correlation (e.g. normalized distance metrics, Pearson and cosine correlation, empirical mutual information) between flows entering and exiting the network [22, 23, 24, 25]. A separate line of work has pursued active flow correlation attacks that insert "watermarks" into network flows – by delaying or dropping packets – that can survive the transformations introduced by various network conditions [26, 27, 28].

While Tor does not attempt to defend against *global* passive adversaries, Feamster and Dingledine [29] introduced the idea of AS-level adversaries and showed that such adversaries could potentially observe the entry and exit flows of a significant fraction of the Tor network. Following this work, many researchers have investigated how routing dynamics and potential manipulation of the routing infrastructure could position an adversary to observe a larger fraction of traffic flows into and out of the Tor network [30, 31, 32, 33, 34, 6, 35, 36, 37], and introduced systems intended to reduce the fraction of potentially observed flows [38, 39, 40, 41].

2.4 Classification

2.4.1 Classification Algorithms

In this section, we briefly discuss support vector machines (SVMs), multilayer perceptron (MLP), convolutional neural network (CNN), which are methods for supervised learning, and autoencoder (AE), which is an unsupervised learning method. In particular, we built our classifiers or correlation methodologies based on SVMs in Chapter 3, MLP, CNN, and AE models in Chapter 4, and CNN models in Chapter 5 and 6.

Support Vector Machines.

Many researchers have used SVMs to construct effective fingerprinting attacks [7, 42, 8]. The SVM algorithm finds the maximum-margin hyperplane in a high dimensional space to which we map our samples, which gives the largest distance to the nearest training-data point for all classes.

In Chapter 3, we used a non-linear classifier with a radial basis function (RBF) and n-fold cross-validation to determine the C and γ leading to the highest accuracy, which are inputs of the RBF. We varied C between 0.0078125 and 128, and γ from 0.03125 to 4. The cross-validation accuracy refers to computing the number of examples in each fold that were correctly classified. In Section 3.4, we ran 10-fold cross validation to avoid over-fitting as well as to compute the overall metrics more correctly during testing. In addition, since the cross validation is the most expensive operation, we parallelized it using multiple python workers supported by the Libsvm library [43]. We used 16 workers when the size of background classes was up to 10,000 and 24 workers when it was more than that.

Multilayer Perceptron.

An MLP [44, 45] is a basic neural network, which is a type of feed forward network. It is also known as a backpropagation algorithm with at least 3 layers. It consists of an input layer, one or more fully-connected hidden layers and an output layer. In the fully-connected layers, all nodes have full connections to all activations in the previous layer. Activation functions are computed using a matrix multiplication, followed by a bias offset.

MLP has two procedures, *forward propagation*, which initialize weights and is a forward pass through multiple layers to produce the output, and *back propagation*, which calculates the errors of the output layer, and then updates the weights layer by layer. A single pass-through is called an *epoch* and consists of multiple *batches*.

We applied the softmax function to the output layer, which is the generalization of the binary logistic regression to the multiclass settings. It takes the vector of arbitrary real values and a vector of values in [0,1], where the sum is 1. This real-valued score is a normalized class probability. Since we used a cross entropy to compute the loss,

we analyzed it as an unnormalized log probability for each class and applied a cross-entropy loss, $E{=}-\sum_i^{nClass} t_i log(y_i)$, where i is a class index, $nClass$ is the total number of classes, t_i is a target probability, and y_i is the output probability. The total loss is computed by the mean of E over all training samples.

In Section 4.4, MLP is used to construct p-FP(M) classifiers in Chapter 4.

Convolutional Neural Network.

A CNN [46] consists of one or more convolutional layers, followed by one or more fully connected layers. The forward propagation runs three series of operations, *convolution*, *pooling*, and *classification*. The convolution operation extracts features from the input by learning the features using small squares of input, called *filters* or *kernels*. That is, we slide filters across the width and height of the input and calculate dot products between entries of the filter and the input to generate a 2D feature map. Sliding different filters over the same feature generates different feature maps. CNN can learn the meaningful patterns using this procedure.

Pooling reduces the dimension of the feature maps and thus the number of parameters and computation in the network. The max pooling layer operates by selecting the max element from the feature map for resizing spatially. Then, the high-level features learned by convolutional and pooling layers are fed into MLP for classification. For the back propagation, it keeps track of the index of max activation so routing the gradient becomes simpler than a general backward pass.

In Section 4.4, CNN is adopted to build p-FP(C) classifiers in Chapter 4and to develop feature embedding networks in Chapter 6.

Autoencoder.

An AE [47] is an unsupervised neural network with two neural networks, an *encoder*, which learns lower-dimensional data abstractions, and a *decoder*, which recovers the original data. It aims to predict the input by using fewer hidden neurons than input nodes to learn as much information as it can about hidden neurons. More specifically, since the number of hidden nodes in each hidden layer is less than the dimension of the original input vector, the network is forced to learn a compressed representation of the input data and then reconstruct the input. Through these procedures, the network can discover the interesting structure of the data.

One advantage with an AE is that at the end of training, weights can lead to the hidden layer, and we can train using certain inputs. Furthermore, when we use other data later, we can reduce the dimensionality using those weights without retraining.

Thus, this approach helps reduce the feature dimensionality for data visualization and reduces the noise in the data. Hidden units in an encoder retain as much information as possible while denoising the data. Moreover, we can elaborate on feature extraction using encoded data through the cost function since we have numerous choice on the cost function and can adjust the weight for each class and sample. This power can be used to reflect certain phenomena in the dataset, eventually leading to more efficient and meaningful data representation. In Section 4.3.2, we focus on the functionality of dimensionality reduction [48] and use an MLP for an *encoder* and a *decoder*, while varying the number of *hidden units* in a hidden layer of the encoder.

To construct a more generative model, variational AE (VAE) was introduced by Kingma and Welling [49] and Rezende *et al.* [50]. Instead of memorizing a fuzzy data structure, it generates latent vectors following Gaussian distribution by forcing a constraint to an encoder. Subsequently, to compute the loss of a VAE, two types of losses must be considered: the error between the input and reconstructed data, and the loss between latent variables and the unit Gaussian, reflected by KL divergence. Training VAE is tricky due to the trade-off between these two different losses. Improving the generalization also promotes the quality of data reconstruction by a decoder.

2.4.2 Binary and Multiclass Classification

Binary Classification.

Binary classification is used to classify instances into one of two possible outcomes. However, since we have a multi-labeled dataset, we investigate two approaches. The first is to train classifiers based on binary-label learning by converting all data to be labeled with 1 or -1. The second is to train classifiers with class labels, and convert all monitored labels output by the classifier to 1, and the unmonitored label to -1. In this setting, when we compute the True Positive Rate (TPR) and False Positive Rate (FPR), we ignore the confusion between monitored traces. In other words, we recognize it as a TP because, although the classifier classifies a monitored trace into a different label

in monitored traces, the attacker is still able to determine that it is a monitored trace. We used both binary-label learning and multi-label learning in the binary classification stage for closed and open world experiments.

If we assume that the censor's goal is to block or detect monitored webpages, the precision indicates how many innocent users are misclassified as visiting "monitored webpages" and recall explains how many guilty users evade detection by being misclassified as visiting "background webpages". If the censor is interested in a particular decision among multiple choices to determine if it is a particular webpage, they should focus on reducing false positives (FPs), which would result in increased precision. If the censor is more interested in determining whether any monitored webpage is visited, they will focus on false negatives (FNs), which are measured by the recall.

Multiclass Classification.

Multiclass classification is used to classify instances into one of multiple possible outcomes. To support the multiclass classification using SVMs, we can reduce the problem to multiple binary classification problems with two different strategies: One-against-one (OAO) and One-against-all (OAA). The OAO approach is more popularly used with SVMs since it is faster than the OAA. OAO classification trains $\frac{k(k-1)}{2}$ classifiers (if we have k labels) per all possible pairs of labels while OAA trains k classifiers since its purpose is to classify a single label against all remaining labels. It is well known that OAA is more accurate than OAO in most cases when we use SVMs [51]. For our experiment, we decided to use OAO, yielding more reasonable computational cost for multi-class classification as other researchers have done [52, 53, 42, 8].

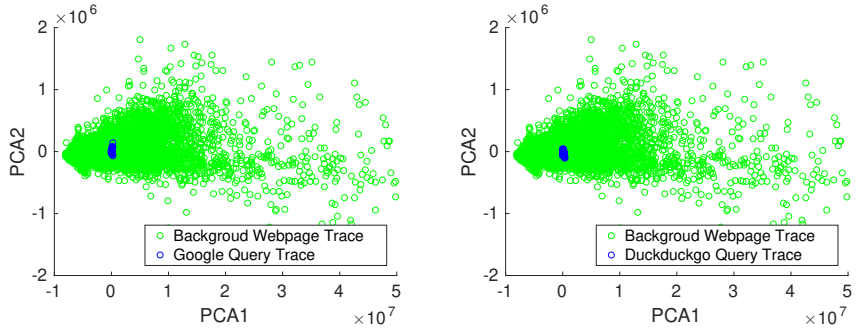
To solve the issue of how to count the confusion between monitored query traces as either FPs or FNs, in Chapter 3, we additionally suggest a new metric, "within-monitored accuracy", which computes the accuracy within monitored query traces. This better represents the probability of the correct classification of individual keywords between monitored query traces. Therefore, in the open-world experiment, we measure success using "within-monitored accuracy."

Chapter 3

Search Query Traffic Analysis

In this chapter, we describe a new application of these attacks on Tor, Keyword Fingerprinting (KF). In this attack model, a local passive adversary attempts to infer a user's search engine queries, based only on analysing traffic intercepted between the client and the entry guard in the Tor network. A KF attack proceeds in three stages. First, the attacker must identify which Tor connections carry the search result traces of a particular search engine against the other webpage traces. The second is to determine whether a target query trace is in a list of "monitored queries" targeted for identification. The third goal is to classify each query trace correctly to predict the query (keyword) that the victim typed.

Note that while KF and WF attacks share some common techniques, KF focuses on the latter stages of this attack, distinguishing between multiple results from a single



(a) Google (b) Duckduckgo

Figure 3.1: Principal Components Analysis (PCA) Plot of Google and Duckduckgo query traces and background webpage traces based on CUMUL feature set

web application, which is challenging for WF techniques. Figure 3.1 illustrates this distinction; it shows the website fingerprints (using the CUMUL features proposed by Panchenko et al. [8]) of 10,000 popular web pages in green, and 10,000 search engine queries in blue. Search engine queries are easy to distinguish from general web traffic, but show less variation between keywords. Hence, while direct application of WF performs the first step very well and can perform adequately in the second stage, it performs poorly for differentiating between monitored keywords. Thus, the different level of application as well as the multi-stage nature of the attack make it difficult to directly use or compare results from the WF setting.

The primary contribution of this chapter is to demonstrate the feasibility of KF across a variety of settings. We collect a large keyword dataset, yielding about 160,000 search query traces for monitored and background keywords. Based on this dataset, we determine that WF features do not carry sufficient information to distinguish between queries, as Figure 3.1 illustrates. Thus we conduct a feature analysis to identify more useful features for KF; adding these features significantly improves the accuracy that can be achieved in the second and third stages of the attack.

Using these new features, we explore the effectiveness of KF, by considering different sizes of "monitored" and "background" keyword sets; both incremental search (e.g. Google Instant[1]) and "high security" search (with JavaScript disabled); across popular search engines — Google, Bing, and Duckduckgo; and with different classifiers, support vector machines (SVM), k-Nearest Neighbor, and random forests (k-fingerprinting). We show that well-known high-overhead WF defenses [15, 16] significantly reduce the success of KF attacks but do not completely prevent them with standard parameters. We examine factors that differ between non-fingerprintable (those with recall of 0%) and fingerprintable keywords. Overall, our results indicate that use of Tor alone may be inadequate to defend the content of users' search engine queries.

[1]Google Instant is a predictive search for potential responses as soon as or before the user types keywords into the search box. By default, Google Instant predictions are enabled only when the computer is fast enough.

3.1 Related Work

Side Channel Attacks and Defenses on Encrypted Network Traffic

The idea of inferring meaningful information based on analyzing encrypted SSL packets was introduced in 1996 [54]. Many studies have exploited side channel leaks in web applications through traffic analysis and investigated their countermeasures. Chen et al. [55] showed that financial information, health profiles, and search queries were leaked over HTTPS and WPA by packet inspection. Schaub et al. [56] and Sharma et al. [57] specifically focused on side channel leaks in Google. Sharma et al. discovered the relationship between typed characters and their exchanged encrypted packet lengths, although this attack has not worked since 2012,[2] and Schaub et al. presented enhanced side channel attacks with stochastic algorithms. For defenses, Zhang et al. [58] developed Sidebuster, based on program analysis to quantify side channel leaks via traffic analysis. Chapman et al. [59] also presented a black-box tool for side channel weakness quantification using the Fisher criterion. Backes et al. [60] adopted a formal approach, enabling information flow analysis to detect side channel attacks.

In contrast, KF targets Tor where all traffic is encrypted, the size of packets is padded to multiples of 512 bytes, and multiple sessions are sent over the same circuit, which makes the traffic patterns less distinct than other encrypted channels. In particular, KF considers broader feature sets beyond traditional feature sets for better classifier performance.

Privacy Protection in Search Engines

Apart from Tor and traffic analysis against defenses, there have been other lines of research on improving privacy guarantee in user search queries based on Private Information Retrieval (PIR) and obfuscating user profiles. Howe et al. [61] and Domingo-Ferrer et al. [62] used bogus queries to mask actual queries and prevent servers from tracing identifiable user information in query logs. Balsa et al. [63] performed an in-depth analysis of privacy properties in web searches and systematically evaluated existing obfuscation-based methods; they found that these methods did not adequately

[2] Since 2012, Google has supported many possible sequences of packet lengths for a given search query to make such prediction more challenging. KF is still effective despite this change, because we did not require a deterministic relationship.

mask a user's actual queries from the search engine. Juarez and Torra [64] proposed a proxy-based approach to dissociate user queries with acceptable overhead in browsing. RePriv [65] proposed in-browser data mining to ensure individual privacy and improving the quality of search by requiring user consent before transferring sensitive information.

3.2 KF Setup

3.2.1 Threat Model

Our attack model for KF follows the standard WF attack model, in which the attacker is a passive attacker who can only observe the communication between the client and the Tor entry guard. He cannot add, drop, modify, decrypt packets, or compromise the entry guard. The attacker has a list of monitored phrases and hopes to identify packet traces in which the user queries a search engine for one of these phrases, which we refer to as *keywords*. We also assume that search engine servers are uncooperative and consequently users relay their queries through Tor to hide the link between previous and future queries. Thus, our threat model only monitors traces entering through the Tor network.

The attacker will progress through two sequential fingerprinting steps. First, he performs website fingerprinting to identify the traffic traces that contain queries to the targeted search engine, rather than other webpage traffic. Traffic not identified by this step is ignored. We refer to the remaining traces, passed to the next step, as *search query traces* or *keyword traces*. Second, the attacker performs KF to predict keywords in query traces. The attacker will classify query traces into individual keywords, creating a multiclass classifier. Depending on the purpose of the attack, the attacker trains classifiers with either binary or multiclass classifications. For example, binary classification will detect monitored keywords against background keywords while multiclass classification will infer which of the monitored keywords the victim queried.

We assume the adversary periodically re-trains the classifiers used in both steps with new training data; since both training and acquiring the data are resource-intensive, we used data gathered over a three-month period to test the generalizability of our results.

3.2.2 Data Collection

We describe the Tor traffic capture tool and data sets collected for our experiment. To collect search query traces on Tor, we used the Juarez et al. Tor Browser Crawler [52, 66], built based on Selenium, Stem, and Dumpcap to automate Tor Browser browsing and packet capture.

We added two additional features to the crawler for our experiments. The first is to identify abnormal Google responses due to rate limiting and rebuild the Tor circuit. When an exit node is rate-limited by Google, it responds (most often) with a CAPTCHA, thus, we added a mechanism to detect these cases and reset the Tor process, resulting in a different exit node.

Among 101,685 HTML files for Google queries and 125,245 HTML files for Bing, we manually inspected those of size less than 100KB, yielding 3,315 HTML responses for Google and 9,985 for Bing, and found that a simple size threshold of 10KB for the response document was sufficient to distinguish rate-limit responses from normal results with perfect accuracy. In contrast, seemingly more sophisticated methods such as testing for the standard response URL (`ipv4.google.com/sorry`) or for an embedded CAPTCHA script (`google.com/recaptcha/api.js`) produced occasional false negatives when a rate-limited node received other abnormal results (such as permission denied).

The second additional feature is to simulate the user's keyword typing in the search box. Google supports auto-complete and Google Instant for faster searches. This can lead to unintentional incremental searches. For instance, when the user loads `www.google.nl` and types "kmart", the result for "kmar" will appear just before the user types 't'. Therefore, to support more realistic user actions, we choose a delay of d seconds uniformly at random from the interval $(0.1, 0.7)$ before typing each letter. Since Google Instant sometimes does not show the result as soon as typing the last letter, we enforce typing RETURN after the last letter is typed. As a consequence, traffic related to suggestion lists, auto-complete, and Google Instant is captured all together with the actual search result traffic.

Since result pages can dynamically modify their content after loading, we wait 5 seconds after the result page is loaded, and then pause 5–10 additional seconds between

queries to capture all traffic related to changes made to the result webpage after loading completes. To collect query traces of monitored keywords, we recorded up to 110 *instances* of each keyword, evenly divided among 6 *batches*. Each batch starts with a list of remaining keyword instances, and repeatedly selects an instance at random from the list, queries the keyword, and removes it from the list until no instances remain. Background keyword traces were collected in 1 batch with 2 instances per keyword. On average, the time gap between batch groups was 3 days.

With these additions and settings in the crawler, we collected several sets of Tor traces to be used in our experiment; they were collected from March to October in 2016. We used the Tor Browser version 4.0.8. (We further collected AOL search query traces from January to February in 2017.)

3.2.3 Keyword Set Details

We used the following three different keyword datasets to investigate the performance of KF across different sets of monitored search queries. Note that we collected "search query traces," labelled them with the corresponding keywords for supervised learning (using the classifiers discussed in Section 2.4.1) and inferred "keywords" based on predicted labels returned by classifiers. For this reason, we refer to KF as keyword fingerprinting rather than search query fingerprinting.

Top-Ranked Keywords.
We harvested 300 *parent* keywords by identifying the top 20 ranked keywords for each alphabet character, a–z, based on Google 's auto-complete [67] (more specifically, `http://keywordtool.io` via Tor). We collected the top 650–750 suggested keywords for each of 100 parent keywords selected randomly from these 300 keywords. This yielded a set of about 80,000 keywords to be used as the background set. For example, if we have a parent keyword, "airline", its suggested keywords returned by auto-complete are "airline pilot" and "airline ticket purchase."

AOL Search Queries.
As another source of monitored and background queries, we downloaded anonymized AOL search logs provided by Dudek [68], and extracted the queries from these logs. The logs are split among 10 files, where each user's queries appear in one log. We randomly

partitioned these logs into two sets, one from which we randomly selected 100 queries to serve as monitored keywords and the other from which we randomly selected 40,000 queries to serve as background keywords.

Google Blacklisted Keywords.
Several previous studies have collected lists of keywords blocked by Google. Among them, we used Google blacklisted keywords reported by www.2600.com website [69] to gather Google blacklisted keywords. For these keywords, searches using Google Instant fail. However, if submitted as normal queries, we were sometimes able to get Google query results for those blacklisted keywords, and sometimes the Google result indicated that the content was blocked. This would allow KF to be used to identify "blacklisted" keywords.

Background sets.
Note that in both the "top-ranked" and AOL data sets, the maximum size of background keyword set we could use for an experiment was 80,000 queries. This is roughly 1–2 seconds worth of worldwide query traffic processed by Google [70], and in line with or larger than the size of background sets used in open-world studies of WF, which generally target coverage of a much broader background space. The use of two differently generated background sets also lends confidence that our results will generalize.

3.2.4 Two Search Query Settings

We considered the following different query settings. Note that we collected Google traces in both settings, Bing traces in the incremental query setting, and Duckduckgo traces in the one-shot query setting.

One-shot Query Setting.
The Tor Browser download page [71] recommends disabling JavaScript (via configuring Noscript(S) to "Forbid scripts globally" for later versions than Tor Browser Bundle 3.5, or "about:config") in order to provide better anonymity and security. To reflect this, we collected traces of the queries introduced above assuming that there is no interaction with a search box. This was achieved for both Duckduckgo and Google by directly requesting a search query url. (e.g., google.com/search?q="keyword" for

Google). Hence, collected traces contain packets for requesting search result HTML and responding with HTML, and requesting embedded web objects in HTML and responding with the corresponding embedded contents. This also mimics the process of typing a query in the search or address bar of Tor Browser when DuckDuckGo is selected as the default search engine.

Incremental Query Setting.
By default, Tor browser enables JavaScript and therefore, traffic related to the user's interaction with the search box has to be captured in addition to all traffic from the previous setting. For Bing, this addition is the traffic related to the suggestion list. For Google, all traffic related to request and response in incremental search results is additionally captured. Including results of both query types allows us to test whether incremental search improves the accuracy of KF.

3.2.5 Data Preparation

We reconstructed full TLS records using `tshark` [72], similar to T. Wang and Goldberg's work [53]. In addition, we removed faulty packets if they were empty, lacked TLS segments, or if the capture file was cut short in the middle of a packet. The latter is a result message from `tshark` when a capture file is not yet flushed out, before a copy is made, in which case the end of the file is not a proper record, since we saved all capture files during a live capture. To correct for occasional out-of-order delivery and re-transmitted packets, we re-ordered all packets according to the TCP sequence number.

3.3 Feature Analysis

After collecting the query traces, we extracted features from the traces to use in classifier experiments. We computed many features previously identified as useful for website fingerprinting, including total number of incoming and outgoing packets and cells, Tor cell traces, rounded TCP and TLS traces, unique packet sizes, outgoing burst data, and cumulative TLS records.

3.3.1 Prior WF Features

Packet and Cell totals (Total).
This is a general feature set widely used in existing work [13, 14, 7, 15]. We computed the total number of packets, total number of incoming packets, total number of outgoing packets, total number of incoming Tor cells, and total number of outgoing Tor cells.

Tor Cell Trace (torCell).
We created a sequence of the number of Tor cells sent in each direction based on the sequence of TLS record sizes. For example, if the sequence of TLS records between the client and guard have sizes 1000, -1500, 700, 500, (where negative numbers indicating incoming packets) the corresponding sequence of Tor Cells is 1, -2, 2 based on the fact that the size of single Tor cell is 512 bytes. We used + to indicate outgoing packets and - to indicate incoming packets.

Rounded TCP (roundedTCP) **and TLS** (roundedTLS).
We rounded the packet size by increments of 600, as Cai *et al.* [42] and T. Wang and Goldberg [53] suggested in their work. These are packet sequences for the rounded size of TCP packets and the rounded size of TLS records.

Unique packets (Unique).
In the Tor network, certain packet sizes frequently appear. We compiled a list of such common packet sizes; in our experiment, we used the range of [-1050,1050], and marked each packet with 1 if it was on the list and with 0 if it was not. This is similar to the work by T. Wang and Goldberg [53].

Burst of outgoing packets .
T. Wang *et al.* [7] introduced bursts of outgoing packets as an identifying feature, where a burst is defined as "a sequence of outgoing packets, in which there are no two adjacent incoming packets." They used statistics of bursts as features; e.g., maximum burst length, number of bursts, etc.

Cumulated TLS records (cumulTLS).
This feature is used by Panchenko *et al.* [8]. First, we extracted a sequence of TLS records' size. If the sequence T=$(p_1, ..., p_N)$ where p_i is the size of TLS record, we

calculated the cumulative sizes, which constitutes C=$(c_1, ..., c_N)$, where c_1=p_1 and c_i=c_{i-1}+p_i. In this project, we only consider the size of TLS records in cumulTLS feature set while they considered size of TCP and Tor cells in their **CUMUL** feature set.

3.3.2 Additional Features

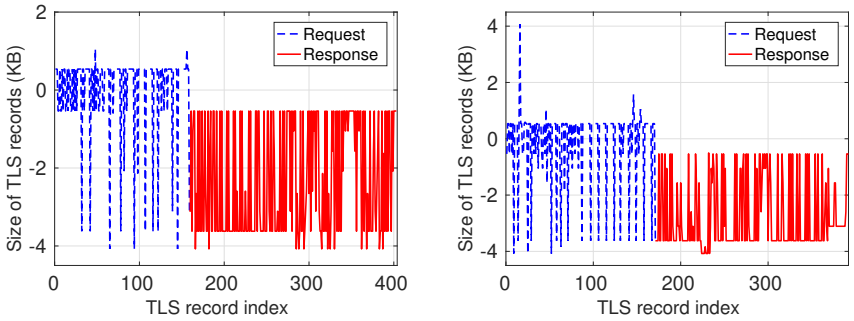
Since search engine result pages often embed CSS, script and advertisement elements, search query traces typically contain multiple request and response pairs; however, the number of such pairs is on average less than those in popular webpages carrying interactive and multimedia content. Additionally, in the incremental setting, after the web browser requests each character in the query, the server responds accordingly to show an updated suggestion list. Subsequently, the web browser requests an HTML document with a keyword and the server responds. Then the web browser requests any embedded web objects such as images. If Google Instant is enabled for Google searches, these interactions are repeated several times. Since the HTML responses are generated by a single programmatic template, the overall timing and size patterns do not have much distinguishing power. As a result, the KF phase will need access to more fine-grained features based on individual TLS records. Thus our experimental results in Section 3.4 consider the following sets of additional features.

Burst of incoming packets.
Based on the concept of "burst" suggested by T. Wang et al. [7], but not exactly following their approach, we defined a burst of incoming packets as a sequence of incoming packets comprising more than 2 incoming packets in which there are no outgoing packets. For each such burst, we computed the total number of packets, mean, maximum, and sum of the TLS record sizes (burstIncoming).

Cumulative TLS data in the response for embedded objects (Resp).
All query traces in our dataset include a giant sequence of incoming packets, which is red in Figure 3.2 and occupies more than 50% of packets in the trace (see Table 3.1). We identify this sequence as the largest incoming burst in a query trace, and call it the "response" portion of the trace, while the sequence before the response portion is the "request" portion.

Table 3.1: Comparison of the request and response portions of Search Query Traces

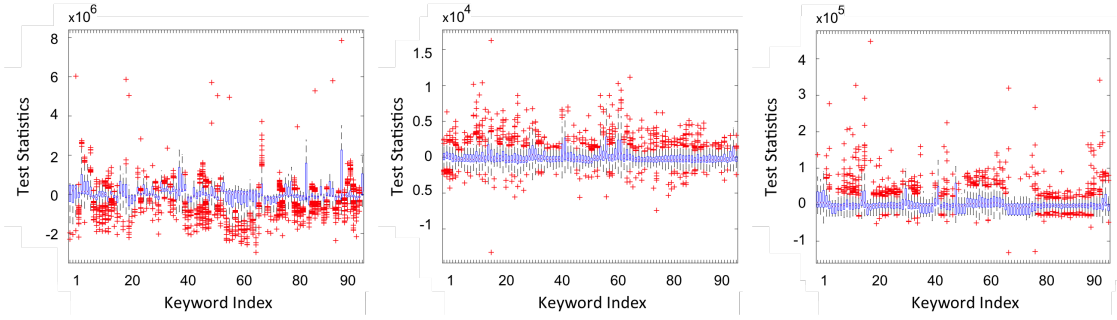
Metric	Google		DuckDuckgo	
	RQ	RP	RQ	RP
Avg of # of packets	140	223	102	193
Max # of packets	288	559	251	801
Avg of total payload(KB)	115	496	89	434
Max of total payload(KB)	350	1,246	295	1,669
SVM Accuracy(%)	13.9	17.2	14.7	20.8



(a) Keyword pizza (b) Keyword craigslist

Figure 3.2: TLS records in two Google query traces. (+) indicates outgoing packets and (-) indicates incoming packets

In the request portion, we are able to capture traffic related to requesting and downloading the HTML response, and requesting the embedded objects. Furthermore, the request portion includes traffic generated by the user's interaction with the search box, e.g. suggestion lists and preliminary HTML results in the case of Google Instant. However, since search query traces from a single web application follow a very similar HTML template and have similar traffic pattern for the interaction with the search box and the predicted search if they are same keywords, we expect this portion of the sequence to be less informative about queries than the response portion. Table 3.1 details results of a small-scale study to confirm this intuition, in which we collected 100 traces for 100 keywords and trained multiclass classifiers on the request and response portions of the traces, respectively. The response portion achieved higher accuracy, 17% compared to 14% of the request portion in Google. When reversing the sequence, as discussed in the next section, the distinction becomes much greater.



(a) Aggr4 (b) roundedTCP (c) cumulTLS

Figure 3.3: Mean Ranks Distribution of Aggr4, roundedTCP, and cumulTLS.

Based on this observation, we extracted feature sets from the response portion: we created the tuple RespTotal consisting of the total number of TLS records, maximum TLS record size, average TLS record size, and sum of TLS record sizes; the sequence of TLS record sizes (RespTLS); the sequence of cumulative sizes of TLS records (cumulRespTLS); and the sequence of the corresponding number of Tor cells (cumulRespTorCell).

For example, if the response portion of a query trace consists of three TLS records of sizes 2080, 3108, and 1566, then the RespTLS feature vector is $(-2080, -3108, -1566)$ (following the convention that sign indicates packet direction). The corresponding cumulRespTLS feature vector is $(-2080, -5188, -6754)$ and cumulRespTorCell is $(4, 10, 13)$. In the following sections, we refer to features extracted from the response portion by prefixing them with Resp; as we show in Section 3.4, feature sets in Resp as well as aggregated feature sets including Resp outperform existing feature sets for the new, second and third stage classification.

3.3.3 Preprocessing

Since both SVM and k-NN classifiers require all input vectors to have the same dimension, we additionally reversed the sequence of cumulative record sizes and tor cells, so that truncation would preserve the end of the sequence, which cumulatively includes information about the earlier portions of the sequence. To show that this improves accuracy versus early truncation, for cumulRespTLS, we computed the accuracy of an SVM

Table 3.2: Sum of squares, mean of squares, and χ^2 of feature sets, returned by Chi-square test statistic, and accuracy of feature sets, evaluated using the SVM classifier over 100 parent keywords. Each feature is described in Sections 3.3.

Feature	SS	MS	H	Acc
roundedTCP	4.5e+10	4.55e+8	1353	12.73
roundedTLS	6.35e+10	6.42e+8	1905	15.16
cumulTLS	7.08e+10	7.15e+8	2123	18.67
Total	2.15e+11	2.17e+9	6461	35.48
burstIncoming	2.8e+11	2.83e+9	8402	26.7
RcumulRespTLS	2.22e+11	2.24e+9	6667	53.79
RcumulRespTorCell	2.17e+11	2.19e+9	6528	53.43

classifier when trained on the first 140 packets of both the original and reversed cumulative traces, for a test set of 100 instances of 100 keywords. As a result, the reversed sequences, RcumulRespTLS, gave us better accuracy (53.79%, compared to 21.33% when using truncated cumulRespTLS). Therefore, we used RcumulRespTLS and RcumulRespTorCell for aggregated feature sets, rather than cumulRespTLS and cumulRespTorCell.

3.3.4 Feature Evaluation

There are several statistical methods to compare the distributions of two sample populations. The Kolmogorov-Smirnov two-sample test decides if two datasets are from the same distribution by comparing their empirical distribution functions and the Mann-Whitney U test supports the comparison of two groups of continuous, non-normally distributed data. In contrast, the the Kruskal-Wallis H Test [73] is a widely used non-parametric technique to test for statistically-significant differences between *multiple* groups of continuous data, using ranks for each feature instead of actual values.

Since comparing features extracted from keyword traces involves comparing more than two groups, we decided to use the Kruskal-Walls H test to determine what set of features to use for classification in the KF phase. We applied it to various combinations of the feature vectors described in the previous section, for a data set consisting of 100 instances of 100 keywords. We ran the Kruskal-Wallis H Test for each feature set, which eventually returns an ANOVA table consisting of sum of squares, degrees of freedom, H, and p-value for each keyword group. Since there are g entries in the ANOVA table,

(where g is the number of keywords in the data set) H is treated as a χ^2 statistic with $g-1$ degrees of freedom to determine the p-value. Given the translation into ranks, H is computed as

$$H = \frac{12}{N(N+1)} \times \Sigma \frac{TR_g^2}{N_g} - 3 \times (N+1) \text{ , where}$$

- N is the total number of features
- TR_g is the rank total for each group
- N_g is the number of features in each group

Because of the translation to rank data, mean ranks of groups are comparable across feature sets. For features that are well-separated, we expect more variation in mean rank across groups (since the features within a group will all have similar rank), and for features that have low distinguishing power we expect the mean ranks of each keyword group to be more similar.

The full results of this test, for the dataset consisting of 100 instances each of 100 keywords, appear in Table 3.2; most of the features gave a p-value of less than 0.01, but H values for some classic features were not as high as for the new features described above. Based on these results, we decided to test different combinations of feature sets whose H (after scaling for differences in dimensionality) was higher than 6,000. As expected, when we included new feature sets Resp and burstIncoming, we found a higher variance between each group. To illustrate this, we computed TR_g^2/N_g, for each keyword group in three feature sets: Aggr4, which combines the Total, cumulRespTLS, RespTotal, and cumulRespTorCell features; roundedTCP; and cumulTLS). As shown in Figure 3.3, it is clear that compared to Figure 3.3a, all keyword groups in Figure 3.3b and Figure 3.3c are close to each other, which prevents differentiating keyword groups based on this feature.

In the end, we found that it is least likely that samples in each group in Aggr4, aggregated based on Total, RespTotal, RcumulRespTLS, and RcumulRespTorCell, are from the same distributions, which leads to better keyword classification results.

Table 3.3: Google trace identification

Background Size	40k	80k	100k
TPR(%)	99.2	98.6	98.6
FPR(%)	0	0	0
precision(%)	100	100	100

Table 3.4: Bing trace identification

Background Size	40k	80k	100k
TPR(%)	99.7	99.9	99.8
FPR(%)	0	0	0
precision(%)	100	100	100

3.3.5 Feature Dimensions

Because of the need to truncate feature vectors to a fixed dimension and because longer feature vectors linearly increase the computational cost of training both k-NN and SVM classifiers, we ran a separate experiment to determine the dimension, n_{best}, that gives the best tradeoff between accuracy and training time, similarly to Panchenko et al. [8].

As Aggr4 was determined to show the highest variation across keywords, we used this feature set to see the relationship between the number of features and corresponding computational cost. To discover n_{best}, we varied the number of features composing Aggr4 by varying the number of features in RcumulRespTLS and RcumulRespTorCell before aggregation since Total and RespTotal have fewer than 5 features. We trained the SVM with 10,000 top-ranked keyword traces and used 24 python workers to parallelize the 10-fold cross validation (Note that we used 24 CPUs and 32GB memory for this analysis). This work is similar to Panchenko et al.'s work [8].

Finally, we decided to use 247 features as it gave the best accuracy as well as acceptable running time, since increasing the dimension of the feature vectors above 250 did not yield better accuracy, while linearly increasing the running time. In addition, note that since training classifiers is an offline process and we use a three-month period training (as discussed in Section 3.6), 188 seconds is acceptable for training.

Table 3.5: Duck trace identification

Background Size	40k	80k	100k
TPR(%)	99.6	99.6	99.6
FPR(%)	0	0	0
precision(%)	100	100	100

3.4 Evaluation

In this section, we evaluate the feasibility of the KF attack through a series of experiments. First, we show that query traces of a targeted search engine can be identified with nearly perfect accuracy against other webpage traces. With the traffic identified as traces of the target search engine, we evaluate KF in both closed-world and open-world settings across several different experimental conditions.

In particular, we investigate the extent to which our new feature sets described in Section 3.3 outperform existing WF feature sets [13, 14, 53, 15, 42, 8] to identify keywords. Furthermore, we focus on investigating whether both identifying monitored keyword traces and differentiating keyword traces from a single search engine are feasible with our new task-specific feature sets, even with 80,000 background keywords. In addition, to achieve this new task, fingerprinting keywords, we consider new variables affecting the performance of classifiers such as different Tor Browser settings and search engines. We also evaluate different classifiers on search query traces to suggest the best method for keyword fingerprinting and evaluate KF under two WF mitigation mechanisms, BuFLO [15] and Tamaraw [16]. Lastly, we further explore the open questions of what factors affect the fingerprintability of keywords.

Throughout this section, we use a variety of metrics to evaluate classifiers:

- **FPR**: The fraction of traces from background keywords classified as monitored keywords.
- **TPR** and **Recall**: The fraction of traces from monitored keywords classified as monitored keywords, a more useful metric in the case that most traces are from the set of monitored keywords.
- **Precision**: The fraction of positive classifications that are correct, a more useful metric in the case that most traces are not from the monitored set.

- **Accuracy**: The overall fraction of traces that are correctly classified.
- **Within-monitored (WM) Accuracy**: The number of monitored keyword traces classified with the correct label over the total number of monitored traces. This metric is only used for multi-class classification, as discussed in Section 2.4.2.

3.4.1 Search Query Trace Identification

We trained SVM classifiers using Panchenko et al.'s cumulTLS features [8], drawing positive examples from our search engine traces and negative examples from 111,884 webpage traces provided by Panchenko et al. [8] without Google, Bing, or Duckduckgo search queries.

The monitored set consisted of 100 instances of each of 100 different keywords, while the size of the background set was varied from 40,000 to 100,000. As shown in Tables 3.3, 3.4, and 3.5 even with 100,000 background queries, we were able to detect query traces with a minimum of 98.6% TPR and 0% FPR. This result is plausible since we are distinguishing one specific class of page from all other traffic, and query responses follow a more restricted format with fewer embedded objects than other webpages. This distinction is illustrated by the PCA plot shown in Figure 3.1. In the following sections, we restrict the input traces to query traces from the target search engine.

3.4.2 Closed World Accuracy

In the closed world scenario, we assume that the victim may query 100 keywords and seek to classify which of those 100 keywords a given Google query trace represents. As other researchers have pointed out [52, 71], the closed world scenario relies on unrealistic assumptions. However, accuracy in the closed-world setting is a minimal requirement for plausibility. We performed multiclass classification with each feature set in Table 3.6, labeling each trace according to its keyword.

We used 10-fold cross-validation — partitioning the traces into 10 folds of 1,000 — to get the best C and γ for each feature set and to ensure that the training sets and testing sets did not overlap. Finally, we obtained 1,000 predictions for each fold (for a total of 10,000 predictions). As shown in Table 3.6, feature sets using information from the Response portion of a query trace outperformed all previously used WF feature

Table 3.6: Closed-world accuracy of various feature sets.

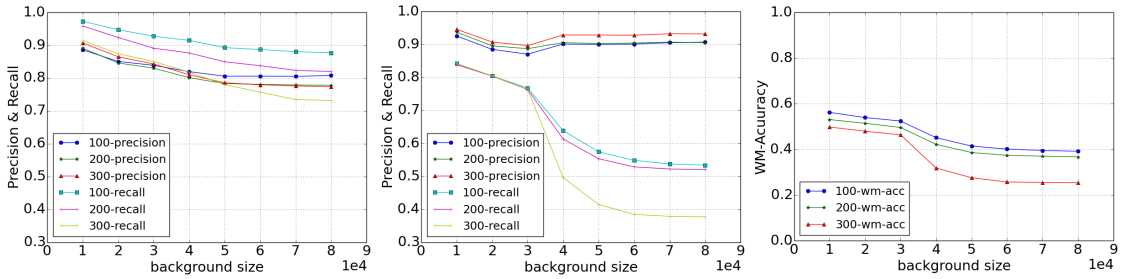
Feature	Accuracy(%)
Total	35.5
torCell	7.5
roundedTCP	12.7
roundedTLS	15.2
burstIncoming	26.7
cumulTLS	18.7
RespTotal	26.1
RespTLS	17.2
RcumulRespTorCell	53.4
RcumulRespTLS	53.8
Aggr2 (Total+RcumulRespTLS)	62.2
Aggr3 (Aggr2+RespTotal)	63.4
Aggr4 (Aggr3+RcumulRespTorCell)	64.0

Table 3.7: Closed-world accuracy (**Acc**), TPR, FPR, and within-monitored accuracy (**WM-acc**) comparing to existing classifiers. (All results in %)

Metric	Acc	TPR	FPR	WMAcc
cumulTLS[8]	18.7	35.0	3.9	8.9
k-FP($k=1$)[9]	40.3	65.4	0.03	35.8
k-NN[7] ($k=1$)	44.5	88.2	22.9	41.1
k-NN ($k=2$)	42.3	32.9	4.70	24.5
k-NN ($k=3$)	43.7	18.7	1.7	16.1
svmResp	64.0	82.6	8.1	56.5

Table 3.8: TPR, FPR, and Precision when we use 100 instances of 100 monitored Google query traces and 1 instance of 10,000 background traces.

Metric	Binary-label	Multi-label
TPR(%)	93.1	82.6
FPR(%)	14.9	8.1
Precision(%)	86.3	91.1



(a) Binary-Label Learning (b) Multi-Label Learning (c) Within-monitored accuracy

Figure 3.4: Precision and recall for binary classification and within-monitored accuracy for multiclass classification when varying the number of monitored and background keywords

sets [13, 14, 53, 15, 42, 8] for KF. In particular, our best feature set (Aggr4) showed much better performance than the feature set (cumulTLS) used for WF by Panchenko et al. [8], achieving a closed-world accuracy of 64.0% compared with only 18.7%. These results clearly show that feature sets tailored to keyword fingerprinting can improve the quality of KF attacks.

3.4.3 Open World Scenario

In the open-world scenario, the attacker maintains a set of monitored keywords to identify, while the victim may query arbitrary keywords. This is a more realistic scenario in the real world, because if the attacker tries to capture victims' search query traces to infer user-typed keywords, the collected data will be expected to include more background keywords than monitored keywords. This section evaluates the performance of several variations of KF in an open-world setting.

Classifier Comparison

First, we compared k-NN [7], CUMUL [8], and k-FP [9] with svmResp to determine the best classifier for KF. In this experiment, we used 100 traces for each of 100 keywords from the top-ranked keyword set, and used 1 trace for each of 10,000 keywords from the background set. We then trained and evaluated classifiers using 10-fold cross validation in the *multi-label learning* setting, where each monitored trace was labeled with its corresponding keyword, and all background traces were assigned a single "background"

label. Note that we used the Aggr4 feature set for k-NN and k-FP.

The results are summarized in Table 3.7. As expected, CumulTLS based on the first 104 features had worse performance than svmResp because the previously used features do not vary enough between query traces from a single search engine. The k-FP classifier was able to differentiate between monitored and un-monitored keywords quite well, achieving a FPR of just 0.03%, but did not do well in identifying the precise monitored keyword for a given query trace. We hypothesize that bagging based on subsets of features discards too much useful sequencing information in the RcumulRespTLS and RcumulRespTorCell feature vectors.

Although k-NN with $k = 1$ had the highest TPR, its FPR was the highest as well. As expected, with higher $k \in \{2, 3\}$, we see a reduced FPR but TPR significantly decreases as well. Note that the FPR is much higher than observed when applying k-NN to website fingerprinting; combined with the observed low FPR of the k-FP classifier, this suggests that the feature vectors of query traces are too densely packed in ℓ_1 space for k-NN to take advantage of the multi-modality of keyword classes. svmResp produced the best within-monitored accuracy rate, which is important in the keyword identification phase of KF. Thus we continue to use svmResp as the classifier for the remainder of the chapter.

Effect of label learning.

We also evaluated KF in the *binary-label learning* setting, where each trace was given a binary label according to whether it belonged to the monitored or background set. As shown in Table 3.8, multi-label learning achieved higher recall (93% vs. 83%) while binary-label learning resulted in better precision (91% vs. 86%).

Furthermore, multi-label learning continues to ensure better precision while binary-label learning results in higher recall across different C and γ pairs. This is because the probability a trace is classified as a FN is lower in binary-label learning since we converted all monitored keyword labels into a single label. Thus classifiers were trained with more instances of the monitored label than those in multi-label learning. For instance, if we select 30 instances for each of 100 monitored keywords, the binary-label classifier is trained with 3,000 instances for a single monitored label. In contrast, the probability a trace is classified as a FP is lower in multi-label learning since the classifier

Table 3.9: Precision(**P**), recall(**R**), and within-monitored accuracy(**W**) (%) to detect 3,000 and 8,000 traces of top-ranked and AOL search keywords and 3,000 traces of Google blacklisted keywords using binary and multiclass classification against 50k–80k background keyword traces. Variance in all figures was less than 0.1%.

Background Size	Top(3,000)			Black(3,000)			AOL(3,000)		
	P	R	W	P	R	W	P	R	W
50k	83	32	22	77	24	24	70	12	12
60k	82	29	20	77	23	23	72	10	9
70k	84	28	19	77	22	22	75	8	8
80k	85	29	20	76	22	22	76	6	6

Background Size	Top(8,000)			AOL(8,000)		
	P	R	W	P	R	W
50k	90	58	41	80	31	29
60k	90	55	40	80	28	26
70k	91	54	40	80	24	23
80k	91	53	39	82	20	19

is trained with fewer instances for each monitored label than in binary-label learning.

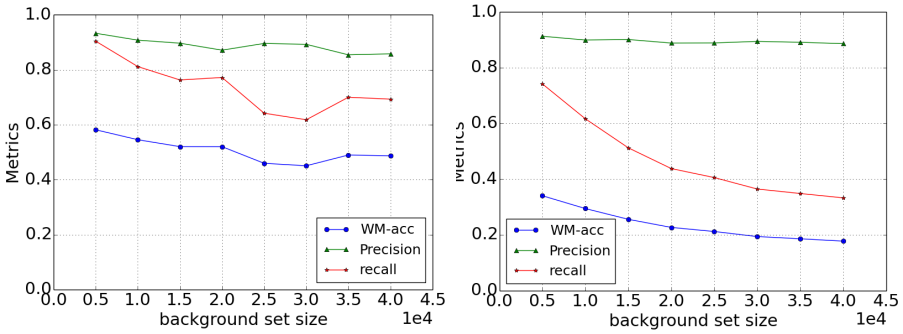
Based on these results, if the attacker wants to correctly identify individual monitored keywords, multi-label learning will have better performance, while also ensuring fewer incidences of censoring non-monitored keywords. If the attacker's goal is to restrict access to keywords in the monitored group, binary-label learning is better since it is more important to ensure that fewer targeted queries evade filtering.

Effect of monitored and background set size.

To determine how the sizes of the monitored and background sets change our results, we selected monitored sets of 100, 200 and 300 keywords, collecting 80 traces for each keyword, and varied the size of the background set between 10,000 and 80,000. In a sense, increasing the size of the background set represents an attempt to capture the large variation in non-monitored search engine queries. Figure 3.4 shows that for binary-label classification, the precision has almost no variation with the size of the monitored set in binary-label learning and minimal variation in multi-label learning, but decreases for both settings with a larger background set. The recall always decreases by increasing the size of either set and is more sensitive to the size of background set in multi-label learning. For multiclass classification, Figure 3.4c shows that increasing the size of either

Table 3.10: Analysis of HTML and search results screenshot for Top-ranked keywords and AOL search queries. We counted the number of content types among text links, images, videos, SNS, and maps, and computed the fraction of HTML responses (t-html), that consist only of text links and include no other contents such as images.

Dataset	HTML size(KB)	# of contents	t-html(%)
Top-ranked	429±131	3.5±0.7	12.0
AOL	384±98	1.2±0.8	63



(a) Incremental Query Setting (b) One Shot Query Setting

Figure 3.5: Within-monitored accuracy, precision, and recall to detect 8,000 top-ranked Google keyword traces when varying Tor browser settings (JS enabled vs. disabled)

set results in a decrease in within-monitored accuracy. However, all of the metrics seem to stabilize with background sets of size 50,000 suggesting these experiments accurately capture the variability in search engine result fingerprints.

Effect of monitored keyword set.

To determine how the set of monitored keywords impacts our results, we further constructed two additional monitored sets using AOL search queries and Google blacklisted keywords. Using 30 instances for each of 100 top-ranked, AOL, and Google blacklisted keywords, Table 3.9 illustrates that KF is more effective for top-ranked keywords. However, with more training data using 80 instances, KF is still able to distinguish between 100 monitored AOL keywords with recall of 31% and WM-accuracy of 29%.

Thus, we can make KF work adequately on different monitored datasets with sufficient training data while the specific set of monitored keywords does somewhat impact the performance of classifiers. To further explore why top-ranked keywords are better

targets for KF, we examined search result screenshots and HTML responses for top-ranked keywords and AOL dataset. According to Table 3.10, we found that top-ranked keyword traces contained more diverse types of contents as well as larger embedded objects than AOL search queries and these impact the performance of KF. In Section 3.5, we furthermore intensively investigate the relationship between specific types of contents and fingerprintability.

Effect of different query setting.

For users who disable JavaScript in Tor Browser as discussed in Section 3.2.2, we also investigated KF in the "one-shot query setting," which includes neither interaction with the search box nor the incremental results returned by Google Instant. We classified 100 instances of 100 top-ranked keywords with varying background set sizes. As Figure 3.5 shows, all metrics except precision in one-shot query traces were worse than those in incremental search query traces. In particular, within-monitored accuracy was significantly lower. (17% vs. 48% for 100 monitored and 40,000 background keywords)

The main reason is that incremental traces carry additional rich information such as traffic for auto-complete and Google Instant search results, which are highly likely to be consistent in the same keyword group. Additional regular traffic makes Aggr4 more distinguishable than features only based on incoming traffic for embedded objects in HTML responses returned by the one-shot query. Thus, the incremental query setting is more vulnerable to KF, indicating that disabling JavaScript also helps to mitigate the KF attack.

Effect of search engine.

We evaluated classifiers for binary and multiclass KF trained using 100 monitored top-ranked keywords and varying background sets across three different search engines, Google (Instant), Bing, and Duckduckgo. As shown in Tables 3.11 and 3.12, for binary classification, TPR is higher with Google as the size of background set increases. For multiclass classification, WM accuracy is consistently highest for Google, due to the extra information leaked by incremental search. Overall, this study shows that our approach can be applicable to most search engines since their query traces follow a similar format, containing a large and informative response portion in their TLS record sequences.

Table 3.11: Binary classification (TPR (%)). We did not report the standard deviation, which is less than 0.1.

Background Size	Google	Bing	Duck
10k	81.2±.1	78.4±.1	75.2±.1
20k	77.2	74.9	71.4±.1
30k	71.8	66.2	60.0±.2
40k	67.2±.1	61.2±.2	56.2±.1

Table 3.12: Multiclass classification (WM-accuracy (%)). We did not report the standard deviation, which is less than 0.1.

Background Size	Google	Bing	Duck
10k	54.5±.1	44.3±.2	44.4±.1
20k	52.0±.1	42.0±.1	41.9±.1
30k	45.1	37.7±.1	35.8±.1
40k	48.2±.1	34.9	33.7

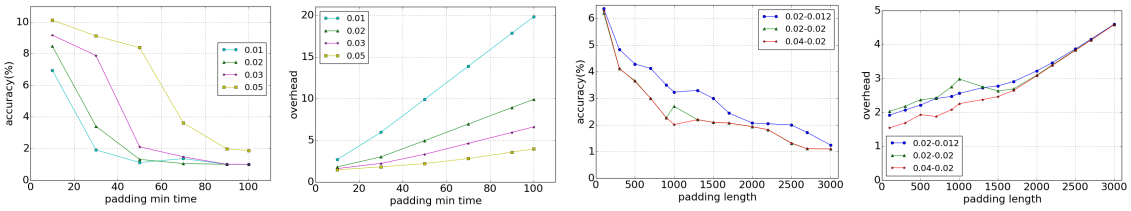
Effect of WF defenses.

We evaluated the effect of two WF defenses on KF in the closed-world setting: BuFLO [15], and Tamaraw [16]. BuFLO enforces packet sizes and inter-packet timing to be constant and pads with dummy packets until the total number of packets reaches a threshold; Tamaraw allows different inter-packet timing for incoming and outgoing traffic and pads incoming and outgoing packets to the nearest multiple of the "padding length" parameter.

We simulated 80 query traces for each of 100 monitored keywords under each defense. However, since both defenses interleave padded outgoing packets with the response portion, the RcumulRespTLS and RcumulRespTorCell feature sequences extracted from these traces were often shorter than the full 120 records used in previous sections: for

Table 3.13: Accuracy (%) under Tamaraw when varying incoming (row) and outgoing (column) padding intervals as well as padding lengths (100–500)

Interval	0.04	0.02	0.01
0.005	5.86±0.25	5.73±0.25	5.89±0.31
0.012	5.24±0.7	5.17±0.88	5.45±0.62
0.02	4.53±0.09	4.66±1.11	5.85±0.89
0.05	6.09±1.2	6.74±0.93	7.39±0.37



(a) Accuracy(%) under BuFLO (b) Bandwidth overhead under BuFLO (c) Accuracy(%) under Tamaraw (d) Bandwidth overhead under Tamaraw

Figure 3.6: Closed-world accuracy for 8,000 WF defense applied Google traces when considering different feature dimensions (Note that with no defense, accuracy is 64.03%)

Tamaraw with padding length 100, only 2,300 traces had at least 40 records, and for BuFLO with 10 seconds of minimum time for padding, only 7800 traces had the full 120 records. For both defenses, if we use larger bandwidth parameters (e.g, 200–3,000 for Tamaraw and 30–100 seconds for BuFLO), fewer than 10 records remain in the response portion of most traces.

For Tamaraw, first, we need to find the effective incoming and outgoing padding intervals, that give the least accuracy. We realized that as shown in Table 3.13, KF works poorly with incoming padding intervals less than 0.02 seconds and chose 3 incoming and outgoing interval pairs, which yielded the worst accuracy, for further investigation of KF performance against Tamaraw.

As shown in Figure 3.6, larger bandwidth parameters (minimum padding time in BuFLO and padding length in Tamaraw), led to lower accuracy as well as higher bandwidth overhead. BuFLO completely defeated KF at the expense of 660% bandwidth overhead (with 0.03 seconds padding interval and 100 seconds for padding time) while the "standard parameters" used by other WF work applied to KF with 100 monitored keyword traces led to accuracy of 10.1% at the expense of 146% overhead. Under Tamaraw, KF became no better than random guessing with 458% bandwidth overhead (0.02 seconds padding interval for both directions and 3,000 for padding length) while performing adequately (6.4% accuracy) with 191% overhead against the parameters used in previous work. This experimental result shows that Tamaraw ensures lower bandwidth overhead compared to BuFLO for perfect defense and existing padding-based defense mechanisms are able to frustrate KF attacks at the cost of bandwidth overhead.

Apart from padding-based defenses, there are other defenses intended to prevent the

Table 3.14: Statistics of traces from best fingerprintable keywords. Note that Avg means traces from all 300 parent keywords (30 instances for each) regardless of fingerprintability.

Metric	Top-FP	Non-FP	Avg
# of outgoing packets	84	71	77
# of tor cells	964	883	873
# of packets in Resp	247	193	180
Cumul payload(KB)	462	420	414

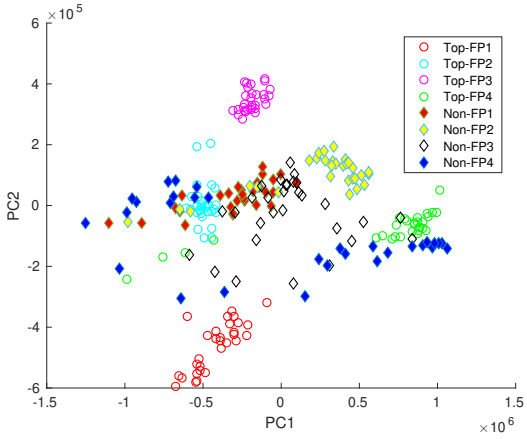


Figure 3.7: PCA plot of best fingerprintable keyword traces and non-fingerprintable keyword traces

search engine from building accurate user search profiles, as described in Section 3.1. We leave the evaluation of KF against those defenses as future work.

3.5 Fingerprintability Analysis

We observed that even in settings where KF had high within-monitored accuracy, some keywords were correctly classified with high probability while others were never correctly classified. To investigate factors that might contribute to fingerprint resistance, we trained a classifier with 30 instances each of 300 monitored keywords and 47000 background keywords from the Google one-shot data set. We selected the 4 "most fingerprintable" keywords (Top-FP, with TPR from 35%–87%) and 4 "non fingerprintable" keywords (Non-FP, TPR of 0%). Table 3.14 gives summary statistics for these

Table 3.15: Analysis of Dynamic and Static contents embedded in HTML

DS	Contents	Non-FP(%)	FP(%)
Dynamic	RHS-DIV	59	45
	News	57	42
	Twitter	17	15
	Stock	11	9
	Reviews	9	5
	People also ask[74]	22	5
	See result about[75]	22	0
	People also search for[76]	22	32
Static	Images	10.8	20
	Video	2	17
	Dictionary	4	32
	UI(e.g.,inner searchbox)	17	26

groups, not showing significant differences outside of the Resp features. We plot them in Figure 3.7, showing that Top-FP keywords are all tightly clustered whereas the Non-FP keywords are more widely spread; however, we found that several Non-FPs not plotted in Figure 3.7 constructed tighter clusters, discussed later.

Therefore, for a more precise comparison, we chose the 33 most fingerprintable keywords (with TPRs from 17% to 87%) and 52 Non-FP keywords to evaluate how several factors contribute to fingerprintability:

Screenshot equivalence.

First, for each of the 85 keyword groups, we checked if screenshots of the instances recorded by the crawler after loading were the same. Second, for groups whose screenshots of instances were the same, we further investigated the contents carried by the HTML responses to identify what specific types of contents vary between the FP and non-FP groups.

We found that 58% of FP keywords and 44% of Non-FP keywords have the same screenshots in all instances. We further analyzed their embedded contents in HTML. Table 3.15 shows that Non-FP keywords generally delivered more dynamic contents[3] than FP keywords. For example, 59% of Non-FP traces include a large right-hand side DIV block — containing various dynamic contents such as stock price and user reviews

[3]The definition of dynamic contents is that the content is variable enough to be changed within 0.2–54.7 hours.

Table 3.16: Unmonitored keyword set analysis

(a) Euclidean distance in PCA plot. (Note that scale is e+5)

Dist	same	other	back
FP	4	7.7	7.8
NFP	3.5	6.3	5.7

(b) Within-monitored accuracy(%) with different background keywords

Ratio	.2	.3	.4
back-c	21	29	34
back-v	47.7	47.9	48

— versus 45% of FP traces. Dynamic contents make traces more inconsistent leading to worse accuracy.

Unmonitored keyword set.

We discovered that while the feature vectors of Top-FP groups were generally tightly clustered in PCA space, the feature vectors of Non-FP groups could either be tightly clustered or more widely spread. Thus, we further computed the average PCA Euclidean distance between instances in the same group and to instances in different groups for each Non-FP and FP keyword and compared them to examine the difference according to fingerprintability.

Table 3.16a shows that average distance between each FP instance and other keyword instances was larger than for Non-FP instances, however, the gap was not high. After investigating confusions, interestingly, most Non-FP keywords were misclassified as "background" labels. Therefore, we re-calculated their average distances to background keyword instances and the distinction became more pronounced. In addition, we trained classifiers with 200 monitored keywords and two different 24,000 background keyword sets, "back-c" and "back-v", and tested each classifoer with the same background set used in Figure 3.4b. Traces in "back-v" are more widely-spread than in "back-c," with an average PCA Euclidean distance between instances in the background group of 1.02e+6 versus 3.12e+5, and an average distance to instances in the monitored group of 9.40e+5 versus 3.07e+6. Table 3.16b shows that the WM accuracy was better with "back-v."

Tor restart/out of order or dropped packets.

We had often received CAPTCHAs during Google trace captures. We examined how often the Tor process had been restarted for each keyword group and further how it affected query results. We found no instances showing a major difference in search

results due to the location difference led by a different exit chosen. (For example, we get different search results for a keyword 'man' when we use Google Italia and Google US.) In addition, 54% of FP keywords received a CAPTCHA during collection and the Tor process had been restarted 60 times on average (35% and 23 times for Non-FP keywords). Furthermore, unstable network conditions can lead to dropped, duplicated, or out of order packets. We got the result that 38% of FP traces had such events, whereas 17% of Non-FP contained those.

These results indicate that both CAPTCHA and unstable network condition did not affect the fingerprintability in our experiment.

Page loading time.
We expected that the diversity and number of contents in the resulting HTML affected the page loading time. However, the page load time did not show significant difference between FP keywords and Non-FP keywords. (0.08 sec vs. 0.06 sec on average) In fact, page loading time is highly affected by the network condition at that time.

3.6 KF Deployment and Mitigation

In the real world, users are likely to visit other webpages and could be involved in other tasks such as listening to music at the same time as using search engines. Furthermore, users do not announce when they are querying a search engine to allow the adversary to begin recording a trace. In addition, search traces and results can be sensitive to when and where users send queries. In this section, we discuss how these issues complicate application of KF to the real world and present strategies that an adversary might deploy to deal with them.

Single query trace identification.
First, we need to split a full Tor connection sequence into sessions. T. Wang and Goldberg [77] proposed "split" strategies based on timing and machine learning to perform this separation under more realistic conditions. Similarly, we can use a pre-defined duration t, for which users pause before moving to another webpage or performing some other action such as clicking a link, to determine the splitting point. Such t can be determined experimentally or empirically.

To ensure a better result, we might additionally use the *Resp* feature sets to characterize a single query session and use existing feature sets to represent other webpage sessions. Even though the degree of difficulty as well as feasibility is not in the scope of this chapter, this machine-learning based approach to identify a query session is viable based on the work of T. Wang and Goldberg [77]. To handle noise, they further proposed adding similar high-bandwidth noise to testing data rather than removing the noise to get better accuracy. A similar technique could clearly be applied to KF attacks.

Caching, location and time effect on search results.
In our experiment, we did not consider the user customization effect since Tor Browser by default disables caching and cookie storage between browser sessions. Therefore, we assumed that the same content always is returned to different users if they type the same keyword. For the location effect, based on our Tor crawler logs, the exit nodes were selected among 43 different countries, which should capture a significant representation of the diversity by exit location seen by typical Tor users. In addition, Google has not publicly reported how often they update search results. As McDonald's blog [78] mentioned, it is presumed to be 4–5 times a year. Juarez et al. [52] reported that WF accuracy goes down to around 0% if the gap between collection of training data and testing data is more than 90 days. Therefore, to ensure good results, training classifiers every 1 to 3 months seems like a fair choice.

Mitigation.
As we explored in Section 3.4, padding-based defenses deteriorate the performance of KF since insertion of outgoing dummy packets makes identifying the response portion more difficult in addition to concealing traditional features. More sophisticated outgoing padding that focuses on the largest incoming burst to make it less distinguishable among different keywords in terms of the length of the burst, in a method similar to the sequence padding used in Walkie-Talkie [79], and the size of TLS in the response portion seem likely to reduce the bandwidth overhead while further diminishing the performance of KF attacks.

Based on the results in Section 3.4, Tor alone does not have enough power to protect privacy in search queries against targeted traffic analysis. To avoid bandwidth overhead from padding approaches, cryptography and obfuscation based techniques (as discussed

in Section 3.1) could be adopted to help conceal the link between users and their search queries, but more work is needed to evaluate the value of these techniques in the context of KF.

Chapter 4

Traffic Analysis with Deep Learning

In this chapter, we explore the applications of DNNs to traffic analysis, including their use in feature extraction and selection, fingerprinting attacks, and fingerprint prediction. We evaluate their performance at these tasks across diverse experimental scenarios using different datasets, classification settings, and attack and communication scenarios not covered in previous work [4, 2].

We summarize our key contributions as follows.

Feature Engineering.
To isolate the benefits of DNNs as feature extractors versus classifiers, we investigate the use of an autoencoder (AE), an unsupervised learning technique, to extract low-dimensional representations of a dataset, in combination with the classifiers used in state-of-the-art WF attacks. We notice that when AE-generated features are fed into state-of-the-art WF attacks, classifiers become more powerful than when using hand-tailored feature sets from recent WF attacks while requiring less computational cost. In particular, even with feature vectors reduced to as few as 40 dimensions, classifier accuracy is still improved. This suggests that AEs can be a powerful technique for feature engineering in new traffic analysis attacks.

Varying Classification Tasks.

We studied the suitability of Multilayer Perceptrons (MLP) and Convolutional Neural Networks (CNN) as p-FP classifiers (we use the names p-FP(M) for MLP and p-FP(C) for CNN) in a wider range of settings than Rimmer et al. [4] and Sirinam et al. [2]: in addition to identifying top Alexa web sites in the closed-world and open-world binary settings, we also study the performance of these architectures in other WF tasks including open-world multi-class classification, search query (keyword) fingerprinting [80], Onion Service fingerprinting, TLS-encrypted website fingerprinting, and WF against four traffic padding schemes – BuFLO [15], Tamaraw [16], WTF-PAD [17] and Walkie-Talkie [79].

We show that for all of these tasks, DNNs can achieve equivalent or better results to those published in the literature. In particular, p-FP(C) based on CNNs is capable of identifying 100 monitored websites against 40,000 unmonitored websites with 94% true positive rate and 0.009% false positive rate, using 30,000 traces of monitored training data. We also show that BuFLO, WTF-PAD, and Walkie-Talkie are less effective against p-FP than previous classifiers, since we successfully conduct multiclass classification on 100 websites with 15%, 57%, and 49% accuracy, respectively.

Predicting Fingerprintability

Say that a website w is *p-Fingerprintable* by classifier c if open-world training with w in the monitored set produces a classifier that correctly identifies at least fraction p of instances of site w. In light of the success of DNN-based classifiers, we revisit the study of Overdorf et al. [81] to study the influence on fingerprintability by these classifiers using feature sets that only focus on elements in HTML documents, such as statistics about links and embedded web content, which we call *HTML features*. To the best of our knowledge, this is the first fingerprintability study using DNNs and focusing only on website design-specific features.

We find that several common features are influential for fingerprintability by both DNN and traditional classifiers, and identify ranges for these features that are common to *less*-fingerprintable sites. These results suggest that website designers interested in helping users avoid WF attacks (or onion service designers interested in protecting the location of their servers) can use the features we idenifty to predict the vulnerability of

content pages and alter the HTML source code of those pages accordingly.

4.1 Related Work

In this section, we review three recent studies of WF that are closely related to our work [81, 2, 4] and discuss the differences between their work and ours.

4.1.1 Automated Website Fingerprinting

In their work on Automated Website Fingerprinting (AWF), Rimmer et al. [4] investigated three DNN architectures – LSTM, CNN, and Stacked Denoising Autoencoder (SDAE) – for WF from Tor network traces. Our work differs from theirs in three aspects: we use a different evaluation methodology, evaluate a wider variety of application scenarios, and investigate the utility of autoencoding for feature extraction *independently* of the use of DNNs for classification.

Although we do not conduct a systematic closed-world study, we demonstrate the multinomial classification ability of DNNs in an open-world evaluation. To show the impact of the monitored dataset, we also evaluate p-FP(M) and p-FP(C) using the AWF and Tor Hidden Service (HS) datasets. As opposed to their work, our open-world evaluation follows the methodology of most other recent work on WF [9, 2, 82, 83, 52], in which the adversary trains WF models using both monitored websites and unmonitored websites, allowing better comparison across classifier models.

Application Scenarios.
We explore a more diverse set of WF tasks using deep learning, including fingerprinting search query traces over Tor [80], Tor traces defended by recent WF defenses [15, 16, 18, 17], TLS-encrypted (non-Tor) traces, and predicting the fingerprintability of websites (Section 4.5).

Autoencoder.
Rimmer et al. use a Stacked Denoising Autoencoder (DAE) architecture for feature extraction, based on the DAE, which is a variant of AEs designed to give better generalization. However, the AWF study did not attempt to isolate the effectiveness of AEs for feature extraction from the effectiveness of DNNs for classification using these

features. We address this gap by investigating the use of AE-extracted feature vectors as input to other classification algorithms. In Section 4.3, we use the low-dimensional feature vectors extracted by an AE to train three state-of-the-art machine learning techniques – SVM, k-NN, and k-FP [9] – and compare the effectiveness of the resulting classifiers with that of CUMUL [8], k-NN [7], and k-FP [9] using their features.

4.1.2 Deep Fingerprinting

Concurrently to our preliminary work, Sirinam et al. [2] showed the importance of the details of the CNN architecture by demonstrating that a more tailored "Deep Fingerprinting" (DF) CNN can achieve very high WF performance against Tor traces, even against lightweight defenses such as WTF-PAD and Walkie-Talkie. The CNN architecture they propose has superior performance to the architectures we evaluated for defended Tor traces, but as with the AWF paper, the DF paper did not explore the same breadth of WF scenarios as our work, nor did it examine the use of DNNs for feature extraction independently of classification. Moreover, their open-world analysis focuses on binary rather than multinomial classification.

In contrast, our experiments consider both binary and multiclass classification; we also explore a more diverse set of DNN models, MLP, CNN, and AE, and evaluate them across diverse fingerprinting scenarios (i.e., search query and TLS-encrypted trace fingerprinting) and datasets (i.e., the AWF and Tor HS datasets).

In addition, we explore different CNN architectures based on 2-2D convolutional layers with Local Response Normalization (LRN) while DF uses 8 1D convolutional layers with batch normalization. It has been shown [84, 85] that LRN handles ReLU [1] neurons, which have unbounded activations, and detects high frequency features with large response properly.

4.1.3 Onionsite Fingerprintability

The "Onionsite Fingerprintability Study" (OFS) of Overdorf et al. [81] used both network-level features based on the onion sites' network traces for state-of-the-art WF

[1] Rectified Linear Units, a fast, non linear function giving x for positive x and 0 for negative x

attacks [9, 8, 7] and site-level features based on HTML files and HTTP headers to determine the best predictors for WF outcomes. Our study has key differences in terms of features considered, methodology, and fingerprintability scoring.

Features and Methodology.
Overdorf et al. studied which of these network-level and site-level features were the most important for WF prediction using the relative difference between inter- and intra-class variance of network-level features and a random forest regressor, respectively.

In contrast, we only use features based on Alexa websites' HTML source code, *design-level* features such as the number of tags and characters in data fields. These features are easy to gather and measure without deploying the site as an onion service and have less variation due to external factors (such as network conditions), which makes the predictions easier to obtain and more stable.

Our goal is to identify which design-level features influence predictability by specific classifiers such as MLPs, CNNs, or k-FP. Thus, our study is in a different setting (open-world vs. closed-world), and it produces features that can be applied directly to a site's HTML source code before deployment.

Fingerprintability Scores.
In order to study what makes a website vulnerable to fingerprinting, we must choose some way to assign a label or "fingerprintability score" to a website. The OFS used the F1 score from an ensemble classifier, combining the precision and recall of three classifiers [9, 8, 7].

We use multiple classifiers – MLP, CNN, k-FP, and SVM – independently and compute the score for each of them. We use the *accuracy* of each classifier for a website, calculated as the fraction of correctly identified instances for each website, as the fingerprintability score; this is more appropriate to our goal of predicting the influence of features on the performance of a single classifier as opposed to the OFS goal of measuring the influence of features on a broad range of classifiers.

4.2 p-FP Overview

In this section, we introduce our adversary model, DNN architectures, metrics to evaluate the performance of our classifiers, hyperparameter tuning to find the optimal parameters to train DNN models, and the datasets used to evaluate our DNN models.

4.2.1 Threat Model

As in all prior work on website fingerprinting over Tor, we assume a network-level, passive adversary who is only able to monitor network traces, sent and received by users. In situations involving Tor traffic, the adversary is only able to observe network traffic between the client and a Tor entry guard, and does not control any other relays or servers involved in the communication.

Our attacker is interested in two types of classification problems. In *binary classification* the goal is to determine whether a captured trace is from a small list of monitored pages. In *multiclass classification*, the adversary additionally predicts which of these monitored pages was visited.

4.2.2 DNN Architectures

We discuss general background on DNN models in Section 2.4.1, and details about the selection of the hyperparameters for our networks in Section 4.2.4.

Experiment Setup.
We used Tensorflow [86] with the TFLearn [87] front end for the implementation of DNN classifiers. We split the dataset into training, validation, and testing datasets with size ratios 54:6:40, which led to better generalization of the trained models and helped to avoid overfitting. To train our models, we built five models for each experiment and selected one model yielding the best performance. We then evaluated this model using 20 different iterations, where each iteration consisted of randomly chosen background instances and monitored samples, and each monitored website had the same number of instances.

We used 32 cores and 256GB of memory for all classifiers and the longest job, which trained a CNN consisting of two 2-D convolutional layers and two fully connected layers,

was finished within five days. With GPUs, this running time would be considerably reduced.

MLP.

The p-FP(M) model consists of one input layer allowing a single vector, two hidden fully connected layers, and one output layer with softmax function. We used L2 regularizations for the first two hidden layers and the dropout between those layers to minimize the impact of overfitting. We chose Stochastic Gradient Descent (SGD) as the optimizer and used the categorical cross entropy for the loss function.

CNN.

The p-FP(C) model is comprised of an input layer accepting input data, two convolutional layers with 128 filters, each followed by a max pooling layer, followed by one hidden fully connected layer and a softmax output layer.

For the input shape, rather than n by n matrix (with original feature vector of size n^2), we constructed a 1 by n matrix, so that the vector of 2,500 features is represented as a 1 x 2500 matrix, which is better suited to WF. We also adjusted the format of filters, using 1 x f rather than f x f filters, because network traffic traces are time series rather than spatial data and we would not expect to find useful patterns in multiple spatial dimensions. We applied L2 regularization for each layer and the categorical cross entropy to compute the loss. Between layers, we used LRN.

AE.

An autoencoder consists of an encoder and a decoder network, each consisting of two fully connected layers. We varied the number of units in the second hidden layer to get various feature dimensions to be evaluated with SVM, k-NN, and k-FP classifiers. To extract AE-encoded features from target traces, we saved a trained model and retrieved weights of the second hidden layer to derive compressed vectors of those inputs. We used the mean squared error to compute the loss. We used a ratio of 45:5:50 for training, validation, and testing datasets, and extracted AE features based on two testing datasets after two iterations.

4.2.3 Metrics

Traditional Metrics.

For the open-world experiments, we used the following metrics, adopted in prior WF work [8, 80, 9].

- True positive rate (TPR): The proportion of positive samples that are predicted as positive.
- False positive rate (FPR): The proportion of negative samples that are mispredicted as positive.
- For both binary and multiclass classification, we trained DNN models using distinct labels for each monitored class plus one additional label for all unmonitored traces. The difference between the settings is whether we count the confusion between monitored classes as a TP. In binary classification, if a monitored trace is assigned *any* of the monitored labels, it is counted as a TP, whereas in the multiclass setting, a monitored trace is only considered a TP if the correct label is predicted. This is consistent with previous WF attack evaluations in the literature [88, 2, 8].
- Bayesian detection rate (BDR): Since reporting accuracy without consideration of the base rate or a prior may have led to bias in early attempts to evaluate WF attacks, we follow the suggestion of Juarez et al. [52] and also report the BDR, computed as $P(M|V) = \frac{P(V|M)P(M)}{P(V|M)P(M)+P(V|U)P(U)}$, where V indicates the event that a webpage is classified as monitored, M is the event that the webpage actually belongs to the monitored set, and U is the event that the page belongs to the unmonitored set, i.e. $P(U) = 1 - P(M)$. $P(V|M)$ is approximated by TPR and $P(V|U)$ is estimated by FPR. We computed BDR in the same way as prior work [9, 52].
- Within-monitored accuracy (WMacc): For KF experiments, we used within-monitored accuracy as proposed by Oh et al. [80] to measure the performance of multiclass classifiers. This is computed by the number of TPs divided by the total number of monitored samples.

Confidence Threshold.

In our DNNs, the output layer returns vectors with prediction probabilities for all labels. Even though the label with the highest probability is selected as a predicted label, if this probability is low, this indicates the classifier has low confidence in its prediction. To avoid using low-confidence predictions, if the highest probability is less than a confidence threshold, we regard this case as being classified as "others". We also varied this confidence threshold to study how this factor affects the performance of DNNs. In Section 4.4, for fair comparison to prior WF attacks [2, 4], we use the Receiver Operating Characteristic (ROC) curve, which is a parametric curve that summarizes the tradeoff between TPR (y-axis) and FPR (x-axis) of a classifier as the confidence threshold varies from 0 to 1.

Top-K Analysis.

The prediction probability vector returned by the output layer enables us to consider other meaningful labels if their probabilities are high enough to be trustworthy even if they are not the highest. Such labels are candidates for the top k list. We studied the performance of these "top k" predictions as k varied from 1 to 5, where if a correct monitored label appears in the top k list, we count it as a TP.

Because of the possibility that the features of Top Alexa websites have a common bias, we treat the appearance of the unmonitored label in the top k list more importantly. In the open world experiments, if the negative (unmonitored) label is included in the top k list and the actual label is positive (monitored), we always treat this as a false negative (FN) even if the top k list includes a true label. For example, if the top 3 list includes monitored sites M_1 and M_2 with probabilities 0.89 and 0.1, respectively, as well as the unmonitored label U with probability 0.01, then even if the true label is M_1, this outcome is coded as FN. We applied top k analysis to evaluate KF attacks (Section 4.4.2) and defended traffic analysis (Section 4.4.4).

4.2.4 Hyperparameter Tuning

Training DNNs requires selecting many different hyper-parameters that can impact the quality of predictions. We used the `hyperopt` library [89] to implement the Tree of Parzen Estimators (TPE) [90], a form of Bayesian optimization, to automate the

Table 4.1: DNN Hyperparameter tuning using HyperOpt (**W**: WF, **WH**: WF with TorHS, **K**: KF, **T**: TLS-encrypted traces, **Full**: Fully-connected layer, **Conv**: Convolutional layer, and **O**: Others) .

DNN	MLP				AE	
HyperParam	Choice			Space	Choice	Space
input dim	W 5000	WH 2500	K 10k	0∼ 10k	5000	0∼ 5000
optimizer	SGD			SGD,Adam	Adam	SGD,Adam
learning rate	0.03			0.001∼0.1	0.001	0.001∼0.1
epoch	≤50			10∼1000	10	10∼1000
batch	≤50			10∼100	256	10∼300
number of Full	4			2∼7	4	2∼7
hidden units	1000∼ 3000			500∼10000	200∼ 300	10∼5000
dropout	0.3∼0.5			0.2∼0.9	-	-
activation	tanh			tanh, relu, sigmoid	relu	tanh, relu

DNN	CNN		
HyperParam	Choice		Space
input dim	T 1600	O 5000	700∼ 5000
optimizer	SGD		SGD,Adam,RMSProp
learning rate	0.05		0.001∼0.1
epoch	40∼100		10∼1000
batch	≤30		28∼128
number of Full	2		2∼7
number of Conv	2		1∼4
hidden units	200∼ 300		500∼10000
dropout	0.8		0.2∼0.9
activation	tanh		tanh, leaky-relu, elu
number of filters	128		4∼200
filter size	12		2∼16
kernel size	10		2∼50

search for optimal hyper-parameters. First, we described the search space, as shown in Table 4.1, and constructed DNNs using the `tflearn` library [87]. Then, DNN models were trained and the prediction results were passed to the optimizer, which selected parameter values for the next iteration to minimize the prediction error. We summarize the search space and chosen values for each DNN model used in our experiments in Table 4.1.

4.2.5 Datasets

We detail how we collected our datasets and categorize the network traffic traces for different experimental scenarios.

Data Collection.
We used a modified version of Tor Browser Crawler (TBC) [91] and Tor version 0.4.0.8 to collect our datasets. Our modified version of TBC reset the Tor process after each website visit, but otherwise used the default options. In addition, we used the `page source` option provided by Selenium and the `Beautifulsoup` library to extract the HTML source code for each downloaded page.

Website Tor traces (WTT).
We collected the WTT dataset with a time gap of one week between batches (We named this dataset WTT-time). Each batch executed one week after the previous batch. For the monitored dataset, we collected 10 different batches where each batch collects 100 traffic instances, HTML source code, and screenshots of each of 100 websites. For the unmonitored dataset, we harvested 10 batches, in which each batch was comprised of 6,000 different Alexa websites ranked between 200 and one million. The collection of this dataset ran from August through December of 2018, which was long enough to capture the impact of dynamic web objects in our collection.

Some instances resulted in abnormal HTML files (for example, due to server errors or network conditions). We noticed that these instances always had HTML files less than 10KB in length, but since some sites also had normal HTML files below this length, we filtered out these failures by manually inspecting the screenshot of any instance with a HTML file of size less than 10KB.

After excluding instances with abnormal capture files or HTML files and sampling

uniformly at random from the 10 batches comprising the remaining instances, we ended up with 300 instances each for 100 websites and one instance of each of 50,000 unmonitored websites. Since we failed to extract HTML files for some websites such as `netflix` since the `Beautifulsoup` library could not parse their HTML document structure, our final set of 100 monitored websites were selected from the Alexa top 150 sites.

Since Tor encapsulates all data into cells of 512 bytes, we extracted Tor cell sequences from the set $\{1, -1\}$, indicating that the client sent one cell or received one cell, respectively, from each website trace. We determined the optimal feature dimension using hyperparameter tuning, as described in Section 4.2.4.

To evaluate our models with other datasets, we also used Wang dataset, provided by Wang et al. [7], the "Tor HS dataset", shared by Hayes and Danezis [9], and the AWF dataset of Rimmer et al. [4].

Keyword Tor Traces (KTT).
For both monitored and background traces, we used Google search query traffic instances [80], which include 100 instances of each of 100 top-ranked monitored keywords and 80,000 unmonitored keywords.

Website SSL (Non Tor) Traces (WST).
To collect TLS-encrypted traces, we build a normal Firefox web browser crawler using Selenium's web driver after removing the Tor settings in TBC [66]. This set consists of 9,000 instances (90 instances each) of the Alexa top 100 websites for a monitored set and 9,000 Alexa websites excluding monitored sites for an unmonitored set.

4.3 Feature Extraction

In this section, we demonstrate that DNNs can be used to perform automated, unsupervised feature extraction even for use with traditional classification algorithms.

4.3.1 Features with Autoencoder

An AE is a widely used unsupervised technique for learning data representations and feature dimensionality reduction, because an encoder network projects the original feature vector into a lower-dimensional representation. This feature compression can both

discover useful features and make classifiers more efficient because their training time often depends on the dimension of the input data. Previously, Nasr *et al.* [92] proposed the use of linear projection algorithms from compressed sensing to enable WF [7] and flow correlation attacks [93] with reduced storage and computation cost, at a slight loss in accuracy. In contrast, we use a deep learning framework to do the dimension reduction, which has previously been shown to be an effective methodology to learn structural data representations more efficiently, compared to compressed sensing [94].

We trained an encoder and a decoder network and captured feature vectors compressed by the second hidden layer of the encoder network. We also varied the number of units from 10-100 in this hidden layer to compress the original features into various low-dimensional representations. We evaluated SVM, k-NN, and k-FP classifiers with encoded features and reproduced state-of-the-art WF attacks to compare their performance. This analysis shows that an AE can learn interesting structure about the data while reducing its dimension.

We additionally tested a variational autoencoder (VAE) [49, 50] and extracted encoded features as explained above; however, we failed to extract meaningful traces. With VAE features, we achieved 2% TPR for multiclass classification and 3% TPR for binary classification. While VAEs perform nicely for datasets where the latent space is continuous, allowing random sampling or interpolation to learn variations on data [95], website traffic instances are less likely to have a reasonable local density.

A DAE is another type of AE, adopted by Rimmer *et al.* [4] to prevent overfitting. By using stochastically corrupted input such as adding noise to the input, a DAE avoids learning the identity function. Since we did not experience significant overfitting with an AE and our generated features make the standard classifiers more effective (Table 4.2), we leave evaluation using a DAE as future work although we acknowledge that it is another powerful method for feature engineering.

4.3.2 Feature Engineering with AEs

We revisited previous state-of-the-art WF attacks [9, 7, 8] and used features learned by an AE (AE features) to train k-NN, SVM, and k-FP classifiers. The feature engineering performed by the authors of these works [9, 8], requires a considerable amount of human effort such as manually inspecting the traffic pattern and experimentally deciding the

Table 4.2: The best performance after 20 iterations (**A**(n): n AE features, TRT: Training time, **DC**: Distance Computation time, **m**: minutes). We empirically selected n yielding the best result.

Data	Wang		WTT-time			
Metrics	TPR	FPR	TPR	FPR	TRT	DC
k-NN	90.2	10.3	91.7	26.6	7.3m	-
A(80)+k-NN	97.9	2.1	93.7	14.9	1.1m	-
k-FP	88	0.5	90.1	5.4	39m	82m
A(100)+k-FP	95.9	1.4	91.3	7.8	1.2m	32m
CUMUL	96.6	9.6	86.9	11.3	84m	-
A(80)+SVM	97.6	1.5	91.4	7.8	87m	-

Table 4.3: Performance of state-of-the-art machine learning algorithms with AE features (**Dim**: feature dimension).

Dim	40		80		100	
	TPR	FPR	TPR	FPR	TPR	FPR
k-NN	92±1	15±1	93±1	15±1	92±1	15±1
SVM	88±1	11±1	91±1	7±1	90±1	7±1
k-FP	90±1	8±1	91±2	7±1	91±1	7.8

optimal dimension of feature vectors. Feature extraction based on AEs offers the potential to automate this process. We used 90 instances of each of 100 monitored websites and 9,000 background websites in Wang dataset [7] and 300 instances of 100 website traces and 20,000 unmonitored instances in WTT-time dataset.

We used the classifier implementations of CUMUL [8] and k-FP [9] directly after removing the feature extraction code. However, since Wang's implementation of k-NN [7] is suited to features based on a Tor cell trace, we implemented k-NN using `sklearn.neighbors` with weight learning that computes the inverse of the distance between neighbors to handle AE features, which may not be in $\{-1, 0, 1\}$, in a better manner. For k-FP, we tuned the number of estimators to have the optimal number of trees finding that dimension 1,000 was best for features suggested by their work [9] and dimension 100 was best for AE features. We used both 1 and 3 for k.

To understand the effect of AE feature dimension on the performance of classifiers, we varied the number of units in the hidden layer, which encoded website traffic vectors, from 10-100. We saved a trained AE model and computed the compressed features on target data using an encoder network by loading the trained model.

Since the dimension of encoded features was significantly lower than the length of

the original input vector, this leads to dimensionality reduction while yielding similar or better performance. As shown in Table 4.3, we found that the dimension of hidden units of an AE did not impact the performance of the machine learning algorithms significantly. Furthermore, the degree of improvement is not proportional to the number of features since with 100 features, k-NN classifiers yielded slightly worse results than with 80 features.

In Table 4.2, we also reproduced the results of state-of-the-art WF attacks [9, 7, 8] using WTT-time and Wang datasets in the open-world setting. For the WTT-time dataset, we alse report Training time (TRT), which includes feature extraction time, along with TPR and FPR to show the effect of dimensionality reduction on the computational cost. Since computing all pairwise hamming distances is the most expensive operation in k-FP, we isolated this time in the table. For both datasets, AE features outperformed all traditional WF features with much lower computational cost.

For Wang dataset, we achieved lower FPR than k-NN [7] and higher TPR than k-FP [9] while completely automating the feature engineering.[2] In comparison to CUMUL [8], we obtained similar results in much shorter training time excluding the feature extraction (39 minutes vs. 4 hours).

For the WTT-time dataset, k-NN requires seven minutes for feature extraction while training and testing an AE takes only one minute to generate its features. k-FP takes 36 minutes for feature extraction and 82 minutes for distance computation while k-FP with AE features reaches similar performance with much lower computational cost. With k increased from 1 to 3, k-FP with 80 AE features yielded 83% TPR and 4% FPR. Compared to CUMUL, since the feature dimension is very similar, where we have 80 features for AE and 100 features for CUMUL, training cost is very similar. However, AE features achieved better performance with 91% TPR and 7.8% FPR.

4.4 Website Classification

In this section, we evaluate the applicability of DNNs to various website fingerprinting attack goals.

[2]For Wang dataset, Panchenko *et al.* [8] reported 89.61% TPR and 10.63% for k-NN and 96.64% TPR and 9.61% FPR for CUMUL (SVM), and Hayes and Danezis [9] presented 87% TPR and 0.9% FPR for k-FP when using 4,500 unmonitored fingerprints.

Table 4.4: WF with p-FP(M) with 30k monitored dataset and varying size of unmonitored sets (TPR(**T**), FPR(**F**), and BDR(**B**) (%), and **H**=Tor HS). Note that all results are based on the confidence threshold tuned to yield higher TPR.

Size	Multiclass			Binary		
	T	F	B	T	F	B
20k	89±1	15±1	90±1	90±1	10±1	94±1
40k	87±1	5±1	90±1	88±1	7±1	93±1
30k(H)	94±2	3	70±1	96±1	0.06	97

Table 4.5: WF with p-FP(C) with 30k monitored dataset and varying size of unmonitored sets (TPR(**T**), FPR(**F**), and BDR(**B**) (%), and **H**=Tor HS) Note that all results are based on the confidence threshold tuned to yield higher TPR

Size	Multiclass			Binary		
	T	F	B	T	F	B
20k	95±1	1±1	97±1	95±1	0.007	98±1
40k	93.77	1±1	98±1	94±1	0.009	98±1
20k(H)	98.49	3.69	78±1	98.91	0.18	98.6

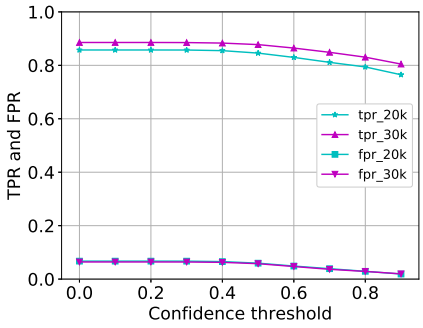
4.4.1 Website Fingerprinting on Tor

First, we explore the effect of various experimental settings on the accuracy of DNNs in WF attacks. Across all experiments, we evaluated both p-FP(M) and p-FP(C) classifiers in the open-world setting.

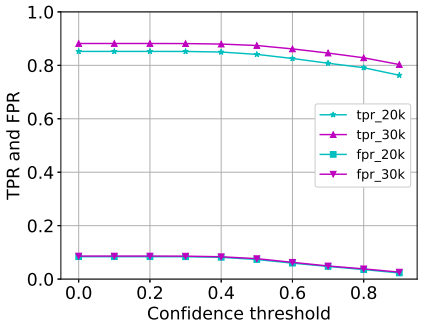
Unmonitored Set.

To show the effect of unmonitored set size, we trained p-FP(M) and p-FP(C) classifiers with the WTT-time dataset, where we have 300 traffic instances of each of 100 monitored websites and one traffic instance each of either 20,000 or 40,000 background websites.

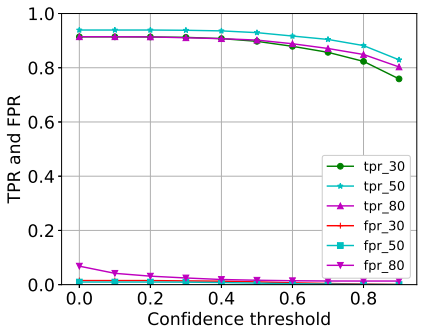
Increasing the number of unmonitored website traffic instances weakened the performance of both classifiers but not significantly, since we measured 94% TPR and 2% FPR for p-FP(C) multiclass classification even against 40,000 background sites, giving BDRs ranging from 97-98% (Table 4.5). Tables 4.4 and 4.5 show that p-FPs are very successful at open-world WF attacks and in particular, p-FP(C) achieved very low FPR in predicting whether a trace was monitored or not.



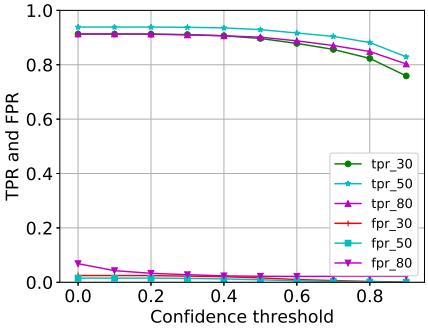
(a) Binary p-FP(M),varying the training data size



(b) Multiclass p-FP(M),varying the training data size



(c) Binary p-FP(C),varying training epochs



(d) Multiclass p-FP(C), varying training epochs

Figure 4.1: WF evaluation using 40k unmonitored set(**a,b**), and using 300 instances of each of 100 monitored sites (**c,d**).

Monitored Set Size.

We also evaluated both p-FP(M) and p-FP(C) when using 200 and 300 instances each of 100 monitored sites. While going from 20,000 monitored instances to 30,000 monitored instances did not have a noticeable effect on the results of p-FP(C), increasing the size of the monitored set somewhat improves the results of p-FP(M) as shown in Figures 4.1a and 4.1b.

Number of Training Epochs.

More epochs improve the quality of classification for both models, as presented in Figure 4.1c and 4.1d. However, after 50 epochs, both TPR and FPR started to decrease. Thus, we chose 50 epochs as the optimal number of epochs to train p-FP(C) models.

Dataset.

To explore the effect of different datasets on the performance of p-FP classifiers, we also used the Tor HS dataset [3] ((H) in Tables 4.4 and 4.5). Both classifiers performed more effectively with 94-98% TPR and 3-4% FPR. These results suggest that p-FPs can achieve very low open-world FPR regardless of the monitored set, but some monitored sets will result in better TPR than others.

Confidence Threshold.

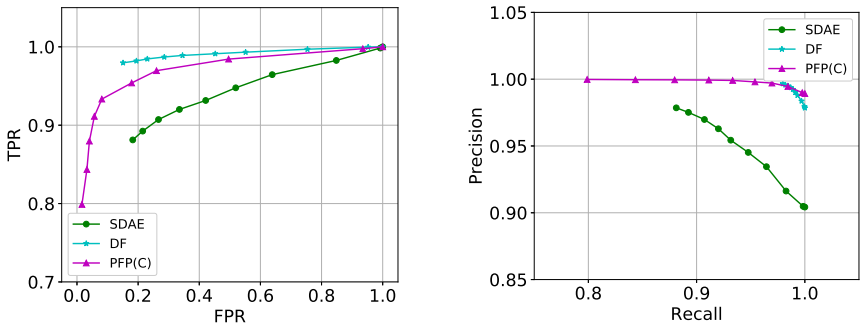
The confidence threshold represents the reliability of decisions by classifiers. We applied confidence thresholds ranging from 0 to 90% (0 corresponds to argmax) to the prediction probabilities of our classifiers, and evaluated both classifiers using both 20,000 and 30,000 monitored traces with 40,000 background traces. As expected, we found that increasing the confidence threshold reduced the number of confident TPs, which lowered TPR, and increased the number of TNs, which decreased FPR (Figure 4.1).

Network Architecture.

As shown in Tables 4.4 and 4.5, p-FP(C) performed much better across all sizes of background sets and both classification tasks. In particular, p-FP(C) yields very low FPR in the WTT-time dataset. More interestingly, p-FP(C) exhibits much better performance for multinomial classification; the number of TPs does not change significantly if we switch the classification task from binary to multiclass classification. In other words,

[3] Comprised of 90 instances each of 30 onion services and 30,000 background website traces

with p-FP(C), the number of confusions between monitored classes decreases.



(a) ROC (b) Precision-Recall Curve

Figure 4.2: Comparison to SDAE and DF using 300 instances of each of 100 websites and 40k unmonitored traces in WTT-time dataset.

Prior Work.

We also trained and evaluated SDAE [4] and DF [2] networks using the WTT-time dataset, to compare their performance. Although Rimmer et al. also evaluated WF using CNN and LSTM networks, we only evaluated their SDAE architecture since it achieved the best performance in open-world experiments in their paper [4]. As shown in Figures 4.2a and 4.2b, DF outperformed SDAE and p-FP(C). p-FP(C) produced very low FPRs (high precision) even for lower confidence thresholds, with FPRs of 0.008% for p-FP(C) vs. 14.29% for DF and 15.88% for SDAE. However, the TPR of p-FP(C) was significantly lower compared to other attacks, with a TPR of 74.84% for p-FP(C) vs. 97.59% for DF and 86.75% for SDAE. This means that with very high confidence thresholds – greater than 0.96 – p-FP(C) predicts that more monitored samples are unmonitored. With a much larger dataset, the degree of drop in TPR would be reduced, however, we leave detailed investigation of the tradeoff between network architecture and required dataset size for future work.

Summary.

p-FP classifiers have been shown to be effective for WF across the different experimental scenarios; compared to related work, although p-FP shows less stable performance than DF, it yields lower FPR across different types of monitored dataset.

4.4.2 Search Query Fingerprinting on Tor

We study the more fine-grained classification ability of DNNs by applying them to KF, which fingerprints Google search query traces over the Tor network. We used 100 instances of each of 100 monitored Google keywords and 10,000 background keyword traffic instances in the KTT dataset; we also used two feature sets, RESP traces (defined below) and Tor cell traces.

Table 4.6: KF with p-FP(M), and p-FP(C) using RESP and cell traces. ((b):binary classification, (m):multiclass classification)

Metrics	RESP(C)	RESP(M)	Cell(M)
TPR(b)	86±1	88±1	59±2
FPR(b)	5	5±1	11±2
BDR(b)	95	95±1	85±2
WMacc(m)	22±1	27±1	22±1
BDR(m)	59±2	63±1	50±1

Features.

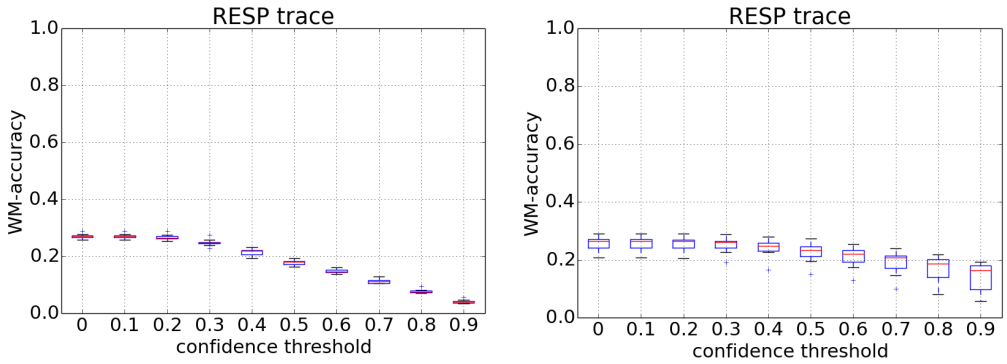
RESP features are extracted from the "response portion" of a Tor trace, defined as the largest sequence of incoming packets, and in previous work, oh2017fingerprinting et al. [80] extracted the sequence of cumulative sizes of TLS records from the response portion. In this work, we ignored the cumulative setting because it gave us lower accuracy.

Since oh2017fingerprinting et al. [80] found that RESP-based features improved the performance of KF attacks using SVMs, we explored whether this is true with p-FPs. Compared to Tor cell traces, Table 4.6 shows that RESP features substantially enhanced the p-FP(M) classifiers' performance. Furthermore, with RESP features, we achieved better performance (88% TPR and 5% FPR) than svmResp [80] in binary classification since svmResp yielded 82.6% TPR and 8.1% FPR. RESP features rather than cell traces help DNNs do better KF classification.

Top-K Analysis.

Compared to binary classification, multinomial classification results using p-FP classifiers were not as powerful, as shown in Table 4.6. To further investigate the performance of multiclass classifiers, we conducted top-k analysis. When using the top 5 analysis,

MLP classifiers achieved higher WMacc than svmRESP [80] (62% vs. 55%).



(a) p-FP(M) (b) p-FP(C)

Figure 4.3: KF(RESP) with p-FP(M) and p-FP(C) by varying the confidence threshold

Confidence Threshold and Network.

As expected, increasing the confidence threshold deteriorates the performance of both classifiers. p-FP(C) also exhibits larger standard deviation with high confidence thresholds than p-FP(M). One possible explanation for this is that since there is much less distinction power between keyword traces, fewer local patterns are learned by filters, making CNNs a less ideal choice for this task. However, this bottleneck can be improved with more training data, which would make the prediction results more consistent across different portions of the dataset.

Summary.

As opposed to general WF, researcher-selected RESP features enhance the performance of p-FP classifiers for KF, and using top-k analysis, p-FP classifiers provide better KF results than prior work [80]. However, there is still room for improvement in KF by discovering different features or training other types of networks, which we leave as future work.

4.4.3 WF with TLS Proxies

We evaluated both p-FP(M) and p-FP(C) classifiers for WF attacks on TLS-encrypted traces using Firefox, to simulate the use of a TLS proxy. We experimented with several

Table 4.7: p-FP(C) performance using sequence of TCP packet sizes(**TC**) and TLS record sizes(**TL**) after being sorted by time (**1**: Top1, **3**: Top3, **T**:TPR(%), **F**:FPR(%)).

	Binary			Multi		
	TC(1)	TL(1)	TL(3)	TC(1)	TL(1)	TL(3)
T	93±1	93±1	96±1	93±1	93±1	96±1
F	4±1	3	1	5±1	4±1	1

trace representations for this task.

Packet Direction.

We extracted the sequence of TCP packet directions from each trace, with entries of -1 for incoming packets, and 1 for outgoing packets; traces shorter than the minimum length were padded with 0 entries. This resulted in 82.9% TPR and 6.8% FPR for binary classification and 82.6% TPR and 8.4% FPR for multiclass classification using p-FP(C).

TCP Packet Sequence.

We also evaluated a representation based on the size and direction of TCP packets, sorted based on transmission time, resulting in the column labelled TC in Table 4.7.

TLS Record Sequence.

We also extracted the sequence of TLS record sizes and directions and sorted it based on the transmission time, resulting in the columns labeled TL in Table 4.7). As shown there, this representation achieved almost the same results as with TCP packet-based features when differentiating between the Alexa top 100 websites.

MLP vs CNN.

We further evaluated p-FP(M) classifiers using all representations discussed above, however, they failed to yield better performance than p-FP(C). Compared to Tor trace WF, p-FP(C) is a much better model to fingerprint TLS-encrypted network traces since CNNs produce stronger representations of the step by step interactions in website downloads based on the local input patterns in TLS-encrypted traffic.

Summary.

Filters in p-FP(C) successfully learn patterns related to the size of TCP packets and TLS

records. However, the performance of p-FP(C) for this task was surprisingly lower than the results against Tor traces due to the use of a smaller dataset. We leave evaluation on larger datasets for future work.

4.4.4 WF on Tor with WF Defenses

We evaluated p-FP classifiers against several recent WF defenses: BuFLO [15], Tamaraw [16], WTF-PAD [17], and Walkie-Talkie [79]. This study helps to understand how automated feature learning by DNNs is impacted by recent WF defenses. We used Wang dataset and the WTT-time dataset consisting of 300 instances of each of 100 websites and applied each defense to those datasets to generate defended network traces. We further evaluated DF [2] on those defended traces for comparison.

Table 4.8: p-FP(M) and p-FP(C) performance against BuFLO(**B**) and Tamaraw(**T**) (**T**: Top n accuracy and all metrics are %) For bandwidth overhead, BuFLO-Wang=217%, Tamaraw-Wang=181%, BuFLO-WTT=179%, and Tamaraw-WTT=175%. Note that for undefended WTT-time dataset, we measured 90% using p-FP(M), 91% using p-FP(C), and 96% using DF, and for undefended Wang dataset, we got 86% using p-FP(M), 92% using p-FP(C), and 96% using DF.

	p-FP(M)		p-FP(C)				DF			
	WTT		Wang		WTT		Wang		WTT	
T	B	T	B	T	B	T	B	T	B	T
1	9	15	16	16±1	17	13	15	7	15	11
2	14	23	19	29±1	22	21±1	24	12	22	18

After hyperparameter tuning, we built the optimal CNN architecture using one convolutional layer, followed by two fully-connected layers, to fingerprint defended traces. Note that it is a different architecture than used in Sections 4.4.1, 4.4.2, and 4.4.3.

BuFLO/Tamaraw.

Due to padding, traffic instances captured under WF defenses result in longer cell traces: for p-FP(M) classifiers and Wang dataset, we used 20,164- and 15,129-dimensional feature vectors for BuFLO and Tamaraw, respectively; for CNN classifiers and Wang dataset, we used 30,000- and 25,000-dimensional feature vectors. For the WTT-time dataset, we always use 10,000-dimensional features in p-FP and 5,000-dimensional features in DF. In Table 4.8, we show the performance of classifiers using Wang and

WTT-time datasets in the closed-world setting, following the analysis of Hayes and Danezis [9].

Table 4.9: Top-1 accuracy of p-FP(C) for varying parameters in BuFLO (minimum padding time τ, interpacket interval ρ (seconds)) and Tamaraw (padding length multiple (**pad**L)) using Wang dataset. Note that **BO** is bandwidth overhead and all numbers are %.

(a) BuFLO

	min. padding time τ (s)			
ρ (s)	BO	10	50	100
0.02	434	15.59	10.53	5.62
0.03	290	18.58	12.15	6.02
0.04	217	20.16	11.98	5.37

(b) Tamaraw

pad$L$	BO	Acc
100	181	15.66
1000	248	5.61
1500	284	3.14

For Wang dataset, as shown in Table 4.8, p-FP and DF classifiers performed much better than other WF attacks against Tamaraw [9]. This demonstrates that padding-based defenses are not able to completely defeat WF using deep learning, which enables more sophisticated and automated feature analysis. All three classifiers performed slightly worse than k-FP [9] against BuFLO, while still significantly outperforming random guessing. In particular, despite padding at a constant rate, both defenses still expose some information about statistics such as the total size of TCP packets, which is learned by DNN models and results in increased accuracy (16-29%) especially against Tamaraw. For both Wang and the WTT-time datasets, Top-1 analysis of p-FP(C) shows slightly better performance than DF.

We also varied the parameters used in BuFLO and Tamaraw to study their impact on the classifiers. For BuFLO, we varied minimum padding time (τ) and packet interval (ρ) of dummy packets in Table 4.9a, and for Tamaraw, we explored different padding multiples (L), with fixed outgoing and incoming padding intervals of 0.04 and 0.012, as shown in Table 4.9b. As expected, longer padding length and padding time decrease the accuracy, however, even with 100 seconds padding time in BuFLO and padding multiple 1500 in Tamaraw, the accuracy of p-FP(C) still exceeds random guessing.

WTF-PAD.

We evaluated p-FP(C) against WTF-PAD using the defended WTT-time dataset after applying WTF-PAD with normal fits [17]. Although we followed the configuration

Table 4.10: The performance of p-FP(C) classifiers against WTF-PAD. For Top k analysis, we chose the confidence threshold yielding optimal accuracy (Bandwidth overhead=37.7%).

	DF		p-FP(C)	
Top$k$	Undef	Def	Undef	Def
Top1	96±1	93	91	57±1
Top2	97±1	95	94	62±1

that yielded the lowest accuracy in that work, this distribution might not be the ideal setting for our dataset since the overhead was lower than in the original work. We leave further investigation of the impact of these settings to future work. According to Sirinam et al. [2], DF achieved accuracy around 91% against WTF-PAD; Table 4.10 shows consistent results with the defended WTT-time dataset as well. DF yielded 93% accuracy while p-FP(C) achieved 57% accuracy. The quality of prediction by both DF and p-FP(C) is much higher than against BuFLO and Tamaraw since WTF-PAD exposes the original sizes of most bursts and these remaining traffic patterns are learned by the CNN's filters.

Table 4.11: The performance of p-FP classifiers against Walkie-Talkie. For Top (**T**) k analysis, we chose the confidence threshold leading optimal accuracy. Note that **Undef** means traces, collected without the defense and **Def** indicates traces, collected under the defense (Bandwidth overhead=24.8%).

	DF		p-FP(M)		p-FP(C)	
Top$k$	Undef	Def	Undef	Def	Undef	Def
T1	86	45±1	81±1	49±1	82±1	48±1
T2	93	66±1	89±1	56±1	83±1	56±1

Walkie-Talkie.

Using the dataset provided by Wang and Goldberg [79], we trained and tested p-FP and DF against Walkie-Talkie. Table 4.11 shows that even though Walkie-Talkie reduced the accuracy of the DF and p-FP classifiers, both classifiers outperform previously known attacks [79]. More surprisingly, both classifiers can reach accuracy of nearly 50% with confidence threshold 0.5. Compared to DF, although p-FP achieved lower accuracy

than DF for undefended traces, p-FP had slightly higher Top-1 accuracy against Walkie-Talkie (49% vs. 45%).

Combined with the WTF-PAD results, these experiments show that defenses based only on concealing burst patterns do not sufficiently mitigate against DNN-based WF attacks, and more research is needed to design light-weight defenses for these attacks.

Summary.
p-FP classifiers perform more effectively against light-weight defenses than against padding-based defended traces. Compared to DF, p-FP classifiers show comparable or slightly better performance against BuFLO, Tamaraw, and Walkie-Talkie while DF is still the most successful attack against WTF-PAD.

4.5 Fingerprintability Prediction

As noted previously, websites exhibit varying levels of fingerprintability; since DNN models are particularly powerful fingerprinting tools, we revisit the question of what features influence fingerprintability by these classifiers. Although Overdorf et al. [81] previously found that the fingerprintability of website traffic instances is primarily affected by the size of websites, our analysis focuses on discovering website design features that impact open-world fingerprintability by DNNs. This analysis may also lead to lighter-weight WF defenses based on safer website design principles.

4.5.1 Dataset and HTML Features

To construct a dataset for FP prediction, we downloaded 29,000 HTML document files from 100 websites ranked in the Alexa top 150, and then extracted the following *HTML features* from these files:

Links and Domains.
First, we fetched all links and extracted features based on the number of links and domains in a site's HTML DOM. In particular, due to popular usage of Content Delivery Networks (CDNs) and cloud platforms, websites contain many links to resources hosted by these services. In order to show the impact of these links, we extracted the number of

links to third party websites. This feature helps understand whether or not downloading web objects (or content) from different web servers makes the network-level traffic pattern more identifiable than when all downloads occur from a single web server.

Tag Paths.

We also extracted features about each site's *tag paths* as a measure of the site's design complexity. We built the tag paths for a site by scanning a site's DOM and iteratively adding a tag to the current path if it was nested. For example, for a document consisting of tags $< html >< body >< a >< /a >< b >< /b >< /body > < /html >$, there are 4 tag paths, $< html >$ (depth=1), $< html >< body >$ (depth=2), $< html >< body >< a >$ (depth=3), and $< html >< body >< b >$ (depth=3). Based on this computation, we extracted numerical features including the number of tags in each tag path, the frequency of increase or decrease in the size of tag paths, and the depth of tag paths. Including these features captures the complexity of HTML document structure and allows us to investigate its impact on the fingerprintability of the website.

Tags and Other Elements.

The size of an HTML DOM as well as the website can be estimated based on features computed from the number of tags, attributes, and comments, and the number of characters and words in `data` and `style` attributes. We extracted these features to validate the relationship between the size of a site's HTML DOM and the fingerprintability of the website.

Embedded Files.

Different types of files are embedded in an HTML DOM and the network traffic associated with fetching those resources may influence the website's vulnerability against WF attacks. Based on the finding that all image and video contents are nested in an `img` tag, we computed the number and the proportion of image and video files and furthermore, we identified specific file extensions to obtain counts and proportions (e.g., jpg, gif, ico, html, etc.). Since these features impact the size of websites, this analysis contributes to a more detailed understanding of the impact of website size on the fingerprintability.

In all, we extracted 62 total features (list in Appendix D in the paper [1]), named *HTML features*, from the DOM of each site and we normalized the data by computing the rank for each feature by looking at each column in the matrix.

For example, if we had 3 instances, [[3,19,10], [7,10,201], [17,7,25]], we converted those features into the rank information, [[1,3,1], [2,2,3], [3,1,2]] and used these vectors as the inputs to classifiers to determine whether a website is fingerprintable. Note that we used the ratio 50:50 for training and testing set and did such partitioning before we compute the rank for each feature in both sets. For instance, after constructing training and testing data, we replaced each non-normalized feature with its rank in the training or testing set, respectively.

4.5.2 Predicting Fingerprintability

We evaluated the how the ability of DNNs to fingerprint a website was influenced by our new task-specific feature set based on a site's HTML DOM.

Fingerprintability Score.

Fingerprintability is a measurement of the vulnerability of a website to fingerprinting over Tor. In this section, our goal is to predict, from its HTML features, whether the Tor network trace of a website w will be fingerprintable by the classifier c. To measure the fingerprintability, we compute the *accuracy* of each website w by training and testing the classifier c using 300 instances each of the 100 top-ranked Alexa web sites, and a single instance each of 20,000 and 40,000 background websites. We used 10 iterations, where each iteration randomly selects training and testing data with the ratio 60:40. Then, based on the WF results of c, we computed the fraction of instances of each site labelled as True Positives as the fingerprintability of the site.

For choices of c, we used k-FP [88], p-FP(M) and p-FP(C), using the same Tor network trace features for all classifiers. If we choose p-FP(C) as c for WF, we derive the FP score by training and testing p-FP(C). If we choose others, we computed the score using that classifier. Thus, we evaluated each classifier independently to show how its fingerprintability was influenced by our features.

Predicting Fingerprintability

After we calculated the score for each w, we labelled the 62-dimensional HTML feature vector for each visit to w, discussed in Section 4.5.1, with either 1 or 0, depending on whether the corresponding website's *accuracy* was greater than a threshold value t_{fp} (in other words, whether the website is at least t_{fp}-*fingerprintable*) or not, respectively.

Table 4.12: Top 15 HTML features based on the Gini importance when using p-FP(M). **(c)** means common top features, which also appear in p-FP(C) top features (Table 4.13).

Rank	Top 15 Features
1	total number of avi files in HTML DOM
2	(c)proportion of ico files in HTML DOM
3	(c)total number of ico files in HTML DOM
4	proportion of image tags over all tag paths
5	(c)total number of links from same domain
6	(c)min depth of tag paths
7	(c)total number of min depth in tag paths
8	(c)std of number of unique tags per a path
9	proportion of html files in HTML DOM
10	median of number of unique tags per a path
11	total number of unique domains in links
12	total number of domains in links
13	proportion of js files in HTML DOM
14	total number of mp3 files in HTML DOM
15	(c)total number of links

For instance, when the threshold is 30%, the websites whose accuracy is greater than 30% were labeled *more fingerprintable* and those with an accuracy less than 30% were labeled *less fingerprintable*.

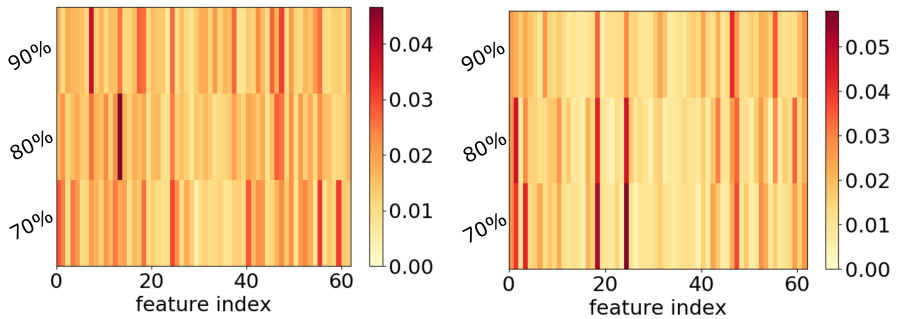
Finally, we attempted to train both MLP and CNN classifiers to *predict* t_{fp}-fingerprintability given the normalized HTML features of a site. We used a $50:50$ ratio to build training and testing HTML datasets, which were randomly sampled every epoch for 50 epochs. Unfortunately, the success of DNN classifiers led to an imbalanced number of instances for each class (fingerprintable and not fingerprintable). Across different FP thresholds, the level of imbalance varies but was always present to some degree; even with threshold 90% only 2,030 out of 14,500 training examples were labelled "less fingerprintable." Furthermore, we closely looked at two groups of HTML feature vectors whose fingerprintability score was less than 90% (*less*) and greater than 90% (*more*). Most feature vectors in the *less* group had accuracy around 80% and the webpage design features in both groups did not have much statistical difference. As a result, we were not able to train a classifier to make useful predictions with this data set.

Table 4.13: Top 15 HTML features based on gini index when using PFP(C). Note that **(c)** means common top features, which appear in PFP(M)'s top 15 features (Table 4.12).

Rank	Top 15 Features
1	(c)total number of links
2	total number of characters in attribute
3	(c)total number of ico files in HTML DOM
4	toal number of css files in HTML DOM
5	portion of jpg files in HTML DOM
6	(c)proportion of ico files in HTML DOM
7	(c)min depth of tag paths
8	(c)total number of links from same domain
9	total number of max depth in tag paths
10	total number of image tags in HTML DOM
11	total number of positive direction in tag paths
12	total number of js files in HTML DOM
13	(c)std of number of unique tags per a path
14	sum of number of unique tags per a path
15	(c)total number of min depth in tag paths

Table 4.14: Most Informative features (**F**) appearing in both p-FP(M) and p-FP(C), and average value of each feature for more fingerprintable (Accuracy $\geq$ 98%), and less fingerprintable (Accuracy $\leq$ 35%) websites. Refer to Appendix D of the paper [1] for the description of each feature index in columns.

F-Index	1	2	11	19	25	47
$\geq$98%	52258	45999	2137	1450	1450	580
$\leq$35%	489427	439476	725	870	870	1749



(a) p-FP(M) (b) p-FP(C)

Figure 4.4: The feature importance of the fingerprintability prediction for p-FP(M) and p-FP(C) with the FP threshold (t_{fp}) 70, 80, and 90%.

Informative Features.

Instead, to gain insight into how more- and less-fingerprintable websites differed in their HTML features, we computed the Gini importance of each HTML feature for each website's t_{fp}-fingerprintability score. Gini importance [96], also called Mean Decrease in Impurity, is widely used as a feature importance measurement. For each feature, the importance score is computed as the sum over the number of splits across all trees in an ensemble that includes that feature. The improvement in each split criterion at each split is the importance score and it is accumulated over all trees for each variable. To investigate which of our design-level features was most informative, we used Random Forests using the scikit-learn library with importance score derived by the total decrease in node impurity, averaged over all trees in the ensemble. Figure 4.4 shows that many of the same important features appeared in common across different accuracy thresholds. To summarize those informative HTML features, we computed the sum of the scores for each feature and selected the top 15 features for predicting fingerprintability by p-FP(M) in Table 4.12 and p-FP(C) in Table 4.13.

There are several top features that appear in predicting fingerprintability by both p-FP(M) and p-FP(C). For each of these features, we accumulated these feature values and averaged them for two groups of sites, one with accuracy less than 35%, and the other with accuracy greater than 98%. The total number of links (feature 1), total number of third-party links and links pointing to webpages within the same domain were the most important features. As shown in Table 4.14, these features indicate that less fingerprintable websites carry more embedded web objects and involve traffic relayed to third party websites, which renders the resulting download traffic pattern less distinguishable. In addition, the structure of the webpage, which is represented by tags and tag paths (features 11, 19, and 25), is also among the most informative features for both classifiers. More fingerprintable websites have more complicated webpage design structure than less fingerprintable ones. All of these top common features highly influence fingerprintability across different FP thresholds and classifiers, as shown in Figure 4.4. These results provide a more detailed view of fingerprintability than the observation by Overdorf et al. [81] that smaller websites are harder to fingerprint.

WF Defenses.

The ability to predict fingerprintability based on document features suggests a new approach to defense for privacy-aware developers. Based on the most informative features for FP prediction, we can guide developers in designing websites more resistant against traffic analysis by suggesting "safe" ranges for these features. This could prevent the website from leaking the web browsing activity of vulnerable users, or help onion services to better conceal their location.

Chapter 5

Data-Limited Traffic Analysis

More recent work has sought to apply methods from Deep Learning to automate the process of selecting classifiers and extracting features from a network trace [97, 1, 4, 2]. By training classifiers on a much larger data set, using raw traffic traces, and applying techniques to search for the best classifiers within a set of models, this work has led to the discovery of classifiers with over 95% accuracy, and which automatically perform feature extraction without requiring prior domain knowledge.

Attaining superior classification performance with these models, however, is only feasible when using a huge training set, such as 800 samples per site. As Juarez et al. [52] pointed out, most websites are regularly updated and modified, requiring the attacker to frequently collect new training traces for each site. This leads to questions about the feasibility of WF attacks for all but the most powerful attackers.

To demonstrate that WF attacks are a serious threat that even less powerful adversaries can use to undermine the privacy of Tor users, researchers have further investigated more advanced deep learning models that can be optimized in scenarios with limited amounts of training data. Var-CNN [11] adapts the ResNet [98] network model with dilated convolution for traffic analysis with relatively smaller training sets. Triplet Fingerprinting (TF) [10] applies triplet networks with N-shot learning to learn a WF feature extractor that can be used for classification with only a few training samples.

In this chapter, we pursue the same objective, *data-limited fingerprinting*, but using generative adversarial networks (GANs) to achieve even better performance for realistic low-data training. Compared to previous work [11, 10], our approach is more efficient,

requiring fewer samples than Var-CNN, and offering better performance than TF without the need for pre-training.

To support training with limited data, we propose the novel GANDaLF architecture, which leverages a small amount of labeled data and a larger unlabeled set to train two networks, a *generator* and a *discriminator*. In our setting, the generator is trained to convert random seeds into fake traces from the same statistical distribution as the training data, while the discriminator is trained to correctly classify the labeled data while also discriminating between real traces and fake traces output by the generator. To the best of our knowledge, this represents the first application of *semi-supervised* learning using a GAN in a WF attack. We show that the GANDaLF approach leverages the generator as an additional source of data to help improve the performance of the discriminator, which enables WF to be effective even in low-data settings.

In addition, our work is the first to examine the performance of low-data WF attacks in a more realistic WF scenario, in which users visit both *index* and *non*-index webpages. We present how the generator effectively generates fake traces resulting in a better discriminator (that is, classifier) even when the *intra*-class variance is much larger than in the standard WF scenario. Note that this second scenario makes GANDaLF more promising, especially when the corpus of site subpages is enormous and requires some limited selection of training subpages. We show that the discriminator can be trained even with very few subpages and work even on previously unseen subpages.

We summarize our contributions as follows:

- ***WF with GANs.*** Ours is the first study to investigate the applicability of semi-supervised learning with GANs to WF models. In this chapter, we introduce GANDaLF, an extension of the SGAN model of Saliman et al. [99] that is optimized for website fingerprinting. Our modifications include adding deeper generator and discriminator networks, applying regularization techniques in a different way, and adopting improved feature matching loss [100]. In addition, we empirically study the choice of hyperparameters and investigate the optimal labeled and unlabeled data settings that result in a more effective WF attack.

- ***Subpage Fingerprinting Study.*** We study the applicability of GANDaLF and other recent data-limited WF attacks to two WF scenarios, WF with index pages

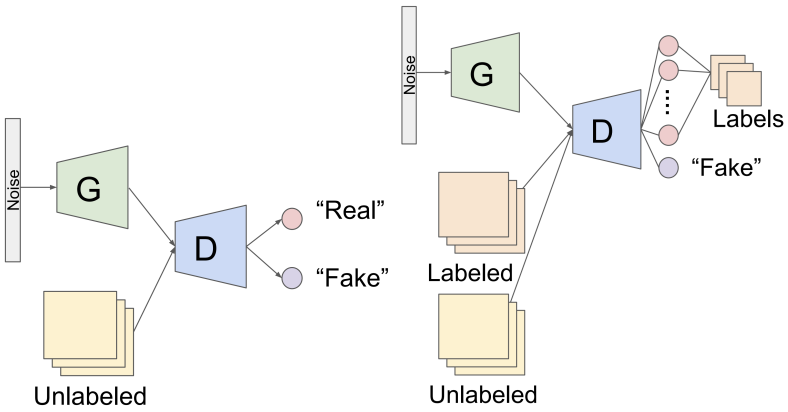
(WF-I) and WF with both index and non-index *subpages* (WF-S). Although other researchers [82, 80] have explored WF-S in the past, to the best of our knowledge, our study is the largest subpage fingerprinting analysis to date and sheds light on the potential of GAN-based fingerprinting. In particular, using the largest website subpage dataset, we revisited state-of-the-art WF classifiers, including traditional machine-learning-based WF, deep-learning-based WF, and more recently published data-limited fingerprinting techniques in both the closed- and open-world scenarios.

- ***Enhanced Low-Data Training.*** In our closed-world lowest data setting using five instances, GANDaLF outperforms Var-CNN [11] by a +40% margin in the WF-I scenario. Even though TF becomes more powerful using five instances in the closed-world WF-I scenario, TF performed poorly in the open-world setting, showing that using the generator of GANDaLF as another data source of "monitored" traces may lead to better generalization in a real-world scenario than the label-based pre-trained network used in TF. Besides the performance improvement, GANDaLF has significant advantages over TF. Unlike the pre-training approach of TF, which still requires regular collection of a large labeled dataset, GANDaLF uses unlabeled data that can often be collected from the same vantage point as used to collect data for the attack itself by running a Tor guard or middle node.

  In the WF-S scenario, GANDaLF outperforms both Var-CNN and TF in all settings by +20-30% and +16-43% margins, respectively. However, we found that k-FP becomes more effective using more limited training data (i.e., 5-20 instances) and in the open-world setting, since feature engineering based on packet statistics is better able to handle large intra-class variance in our subpage dataset. We show that manual feature engineering is helpful in WF-S, while GANDaLF outperforms all other DL-based classifiers.

5.1 Related Work and Bakground

In this section, we discuss recent WF models based on advanced DNN techniques that specialize in maximizing performance in low-data scenarios and GANs we leveraged to develop GANDaLF.



(a) Vanilla GANs

(b) Semi-supervised learning with GANs

Figure 5.1: Generative adversarial networks (GANs).

5.1.1 Low-Data WF

The AWF and DF models were trained using hundreds of samples for every website in the monitored set. Sirinam et al. noted that collecting such large datasets would require an attacker to run between 8-24 PCs continuously [10], where the dataset needs to be refreshed every few weeks at least to maintain high accuracy [52, 3]. Given this, one can criticize the attacks as being unrealistic except for powerful adversaries who could use the resources to instead perform other attacks on Tor. To address this, researchers have begun focusing on the design of WF attacks that overcome the cost and time to obtain training data. Two recent works [11, 10] focus on the optimization of DNN architectures to achieve state-of-the-art performance when few fresh samples are available for training. These attacks allow hypothetical adversaries to quickly apply ready-to-use models, even when subjected to such constraints.

Var-CNN.

To achieve comparable classification performance in low-data scenarios, Bhat et al. [11] introduced Var-CNN, an optimized DNN architecture based on ResNet [98] with dilated causal convolutions. They also proposed to combine direction and timing information together. Those improvements enabled their model to get 97.8% accuracy using just 100 training instances per website across 100 websites.

Triplet Fingerprinting.

To achieve high performance with as few fresh training samples as possible, Sirinam et al. [10] pursued the concept of *N-shot learning* with their Triplet Fingerprinting (TF) attack. For this attack, the adversary first pre-trains a feature extractor using large, publicly available training sets such as the AWF dataset [3]. The feature extractor is trained to produce a vector as output that captures the feature information in a way that minimizes the distances between traces from the same website and maximizes the distances between traces from different websites. Using the features derived from this feature extractor, a small dataset of fresh samples are then used to train and test a simple distance-based classifier, such as k-NN. This attack was shown to reach 94% accuracy when only using 10 instances per website to train the classifier.

Although Sirinam et al. addressed the problem of data-limited fingerprinting, it requires pre-training a model using a large labeled training dataset. While such datasets are available, such as the AWF dataset collected in 2016, Sirinam et al. found a significant loss in performance when using data from three years before [10]. It is likely that the attacker would still need to regularly update the labeled dataset to maintain a high performance level. In contrast, GANDaLF uses a large unlabeled dataset. This dataset can be obtained by eavesdropping on potential victims *from the same vantage points as used to perform the attack*. For example, the attacker could run a Tor guard or middle node or use malware to compromise the DSL modems through which users connect to the Internet. From these vantage points, they can collect unlabeled data to train the GAN as well as the data for performing the actual WF attack, without significant additional cost. As such, to properly update WF models using a fresh dataset, the use of unlabeled data can make WF attacks more portable while keeping high performance.

5.1.2 WF with Subpages

For the majority of the experiments performed in the aforementioned works, researchers have used a website's *index* page to represent the site (e.g. going to `http://www.cnn.com/` by typing it into the browser's URL bar). This attack strategy limits the applicability of the attack as it severely limits the amount of traffic instances that may be viably fingerprinted in the real-world. Panchenko et al. were the first to identify this issue [82]. An attacker would be interested to instead identify a website using traffic

samples generated from *any* webpage on the site (e.g. news stories on CNN.com reached by following links from the homepage or social media). In this scenario, the attacker can use both index pages and *non-index* pages (also referred to as *subpages*) for website identification. To distinguish between these settings, we use the phrases *WF with index pages* (*WF-I*) and *WF with subpages* (*WF-S*).

When Panchenko et al. first studied this scenario, they used a small dataset of 20 websites, each represented by 51 subpages, for which up to 15 samples were collected (i.e. 20x51x15). In this chapter, we extend this study further by collecting a larger dataset of size 24x90x90 and perform a thorough comparison of WF attacks in this setting.

Tangentially, Oh et al. studied the fingerprintability of keyword search queries for popular search engines when visited over the Tor network [101]. This work is similar to that of WF-S, since the webpage generated by each keyword search query represents a unique subpage of the search engine website domain. Our work is different, however, in that each search term's subpages are identified as their own class, allowing for the identification of individual pages within a domain. For non-search cases, this would be equivalent to trying to classify the subpages separately (e.g. https://www.cnn.com/politics as different from https://www.cnn.com/business) instead of as members of the same class (e.g. for CNN.com in general). We use the term *subpage identification* to describe this type of attack scenario. For this chapter, we have limited our study to WF-S and do not explore subpage identification.

5.1.3 Generative Adversarial Networks

Generative Adversarial Networks (GANs) were first introduced in 2014 by Goodfellow et al. for creating images [102]. Figure 5.1a depicts a vanilla GAN, which consists of two components: a *discriminator* and a *generator*. The discriminator is a two-class classifier that is trained to distinguish between real data samples and fake data samples that are generated by the generator. The generator is trained to create those fake data samples in a way that is as hard to distinguish from the real samples as possible. If both models are successfully trained, the generator will become good at producing samples that are indeed very similar to the original data distribution, such as realistic-looking images.

The discriminator uses a mostly standard convolutional neural network (CNN) that

takes the image as input and outputs a single value in the range $[0, 1)$ that can be interpreted as a measure of how real the input image is, i.e. the likelihood that it comes from the distribution of real data, $p_{data}(x)$. The generator, on the other hand, is given a noise vector $p_z(z)$ as input and effectively operates as a reverse CNN, repeatedly using a method called *deconvolution* to transform the noise into image features and create an output in the shape of an image.

The vanilla GAN thus alternately minimizes two different loss functions to achieve these two different goals:

$$\min_G \max_D \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1\text{-}D(G(z)))]$$

In words, the discriminator, D, is trained to maximize the probability of assigning the correct label to both original and reconstructed samples from the generator, G, while G is trained to minimize $\log(1 - D(G(z)))$.

5.2 Datasets

In this section, we discuss the details of the datasets used for our attack evaluation in Section 5.5. The datasets we use are organized into two different types: index-page only data for use in WF-I, and datasets containing a mix of subpages to be used for WF-S.

5.2.1 Index Webpage Set

For the WF-I scenario, we use the large datasets collected by Rimmer et al. in 2016 [3]. To the best of our knowledge, they provided the largest dataset, comprising 2,500 instances of each of 900 monitored websites and 400,000 unmonitored websites. The sites were chosen among top Alexa websites [103].

To produce the dataset used for our evaluations, we chose 200 websites randomly sampled from their monitored dataset. We further partitioned this data into two distinct sets containing samples for 100 websites each, which we denote **AWF1** and **AWF2**. For our WF-I experiments, we use AWF1 as our monitored dataset and use the AWF unmonitored dataset (**AWF-OW**) as is. We then use the AWF2 dataset to act as the unlabeled dataset when training GANDaLF. This is also the dataset we use to pretrain the TF attack. Furthermore, to investigate the impact of labeled data and unlabeled data on the performance of GANDaLF , we adopt DF set (**DF**) provided by Sirinam

Table 5.1: The data setup for GANDaLF in Section 5.5 (Note that **( )**: the number of instances per class, **L**: labeled set, **U**: unlabeled set, **GF**: GDLF, **n**: $\{5, 10, 20, 50, 90\}$).

	WF-I		WF-S	
	CW	OW	CW	OW
L	AWF1 (n)	AWF1&AWF-OW (n&100×n)	GF25 (n)	GF25&GF-OW (n&25×n)
U	AWF2 (2500)	AWF2&AWF-OW (2500&33k)	AWF1 (2490)	AWF1&AWF-OW (2490&400k)

et al. [2] that was collected using 40 circuits and labelled with the circuit index along with the website class.

All traces in AWF and DF set consisted of the packet direction information in which each column was marked as -1 if it is an incoming packet and 1 otherwise.

5.2.2 Subpage Set

In order to evaluate the effectiveness of subpage fingerprinting we needed to collect a new dataset using Tor Browser Crawler [91]. Before beginning the data crawl, we first harvested a list of non-index URLs for websites sampled from the Alexa top 200 rankings.

We then downloaded those websites locally using `torsocks wget` and identified candidate URLs using the `find` command. Finally, we eliminated domains that had low subpage counts and trimmed our final list of URLs to subpages within 25 domains.

Next, we randomly selected an equal number of non-index pages for each website, which we visited many times in a round-robin fashion. After collecting the traffic for each visit, we filtered out traces that failed to load and that contained less than 150 packets. We then trimmed the dataset such that the number of non-index pages and samples per non-index page were equal for every examined website. This process left us with 39 instances per subpage and 96 subpages per website, for a total of 3,744 instances for each of our 25 websites. We will refer to this dataset as **GDLF25** from here on out.

For WF-S CW experiments, we use all 25 domains in GDLF25 as our monitored dataset. To evaluate WF-S in the OW setting, we further crawled the unmonitored subpage dataset using urls provided by other researchers [3], leading to 70,000 subpages (**GDLF-OW**). Finally, we use the AWF1 dataset to act as the unlabeled and pretraining

data for the GANDaLF and TF attacks. Table 5.1 summarizes the dataset usage for two different scenarios: WF-I and WF-S.

5.3 Semi-Supervised Learning with GANs

Given that GANs have achieved success for various *generative* applications, researchers have become interested to see if they can also help in *classification* tasks as well. In particular, GANs have been applied to the problem of *semi-supervised learning (SSL)* [104, 105]. The goal of SSL with GANs is to transform the discriminator into a multi-class classifier that learns from a relatively small amount of labeled data (the supervised part) while also learning from a larger body of unlabeled data (the unsupervised part). As shown in Figure 5.1b, this classifier (the discriminator D) outputs not only a single value that measures whether the input was real (closer to 1) or fake, but also it outputs a set of K individual class probabilities that can be used to identify the label for any real samples. The generator G is trained as before only to produce samples that appear real without necessarily fitting any of the classes in particular, which eventually helps classify the dataset.

The benefit of this approach is that it can achieve high performance on the classification task while only using a relatively small sample of labeled data. In many tasks, unlabeled data is readily available, but obtaining large quantities of data with accurate labels is often expensive. The insight of using a GAN is that, during the process of learning how to discriminate real samples from fake ones, the discriminator is also learning how to extract meaningful features from the data by utilizing a large amount of unlabeled data. Having built up this feature extraction capability, the model is then able to learn how to distinguish between different sites with relatively few samples. Indeed, Saliman et al. [99] showed the potential of this idea by improving the performance of a classification task using GANs in this way. In this chapter, the overall GANDaLF design is based on their GAN, which is called SGAN. We next provide a brief overview of the SGAN design.

5.3.1 SGAN Overview

SGAN is composed of a generator and a discriminator, which are trained together in the same way as a vanilla GAN. The key difference from the vanilla GAN is in the loss functions of the generator and discriminator. Saliman et al. [99] introduced a new loss function, called *feature matching loss*, to be used as the *generator loss*. We will discuss this loss function in detail in Section 5.3.2. The *discriminator loss* is a combination of the *supervised loss*, based on how well it classifies labeled inputs into the K classes, and *unsupervised loss*, based on how well it can distinguish real inputs from the unlabeled dataset and the fake inputs from the generator.

The original concept of the supervised loss was to optimize the classification over $K + 1$ labels where K is the number of classes in the labeled set, and there is one additional label for fake data. However, since the classifier with K + 1 softmax outputs is overparameterized, as an efficient implementation [106], Saliman et al. suggested to fix the unnormalized logit as 0 for fake data, which in turn, leads the supervised loss to a categorical cross-entropy loss over the softmax output of K classes and the discriminator D to be $D(x) = \frac{Z(x)}{Z(x) + \exp(l_{fake}(x))} = \frac{Z(x)}{Z(x)+1}$, where $Z(x) = \sum_{k=1}^{K} \exp(l_k(x))$ where l is the output logits.

5.3.2 Feature Matching Loss

State-of-the-art GANs have been shown to generate excellent samples for many tasks [107], but training GANs is difficult and very sensitive to the hyperparameter settings. A particular issue is that the two models can end up focusing too much on mistakes the other model is making without actually improving on the core task. For example, if the generator is adding a blue blob to the corner of nearly every image, then the discriminator will heavily emphasize this blob in its decisions, improving its score without improving its function. The generator in turn may learn to change the color of the blob to red without removing it. This can create a fruitless back-and-forth that does not yield good images.

To solve this problem, Saliman et al. [99] proposed a new loss function for the generator, called feature matching loss, to prevent it from overtraining on the current discriminator. Feature matching loss is computed on the output of an intermediate layer

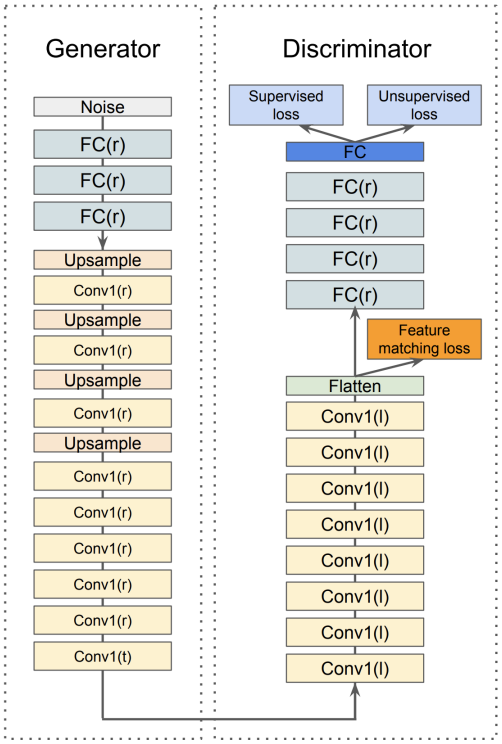


Figure 5.2: GANDaLF architecture (**FC**: Fully-connected layer, **Conv**: convolutional layer, **r**: ReLU, **t**: Tanh, and **l**: LeakyReLU). Note that in WF-I, we used one fully connected layer for the generator.

of the discriminator, which can be thought of as a representation of the features that the convolutional layers have found. By seeking to minimize the difference between the features found in real data and the features found in its generated samples, the generator can avoid focusing unduly on a single mistake it is making that would be heavily emphasized in the final discriminator output. This strategy makes GANs more stable in training, and we leverage it in GANDaLF as well. In GANDaLF, the feature matching loss was computed for both WF-I and WF-S scenarios by capturing the feature vectors in the flatten layer prior to the last convolutional 1D layer, as shown in Figure 5.2.

5.4 GANDaLF

In this section, we introduce our new attack, GANDaLF, starting with the threat model, then a discussion of the intuition behind the choice of SGAN for WF, and present details of its design and network architecture. Furthermore, we emphasize the contribution of GANDaLF based on our experiences and findings when tuning the GAN for WF in the semi-supervised setting and in comparison to prior WF attacks.

5.4.1 Threat Model

We assume a network-level, passive adversary who can only observe the network traces between the client and the middle node of a Tor circuit, possibly by operating the entry guard or middle node. The attacker is not able to drop or modify packets that have been sent and received from servers or collude with web servers.

The attacker requires training data, but since collecting data from a client-based web crawl can be expensive, this attacker seeks to run an entry guard or middle node to both perform the attack and simultaneously gather live Tor traffic that can be used as unlabeled data. Note that while the middle node position has less direct information about the client, Jansen et al. [108] showed how an attacker can use the middle node to perform attacks such as WF.

The attacker is interested in two attack scenarios, both of which we explore in this chapter. The first goal is to train GANDaLF using website index pages and then fingerprint visits to index pages only. This approach has been used by most previous WF research [2, 3, 1, 88]. We call this scenario WF-I, referring to fingerprinting websites with index pages.

The attacker's second goal, which might include more realistic scenarios, is to train GANDaLF using both index and *subpages* (i.e. non-index pages) from a website and then try to identify visits to any subpage of a website. For example, the attacker may want to classify any page of `amazon.com` as Amazon. Note that the attacker only needs a subset of the subpages from each website instead of all pages to train the model, and can test it using unseen subpages by leveraging the generative ability of GANDaLF. We call this scenario WF-S, referring to fingerprinting websites with index and non-index pages.

We explore WF-I and WF-S scenarios in both closed-world (CW) and open-world (OW) settings. In CW experiments, the attacker keeps a webpage fingerprint database and assumes that users will only visit webpages in this database. In the more realistic OW setting, the attacker keeps a set of *monitored sites* and attempts to classify whether a particular trace is to a site in this set or outside the monitored set. To achieve this, the attacker collects traces of both monitored and unmonitored websites to train the classifier and predicts unseen webpages using this trained model to answer whether or not they are monitored.

5.4.2 Sources of Unlabeled Data

Several groups of WF researchers [3, 2, 11, 1] have demonstrated the effectiveness of convolutional neural networks (CNNs) to model the distribution of website traces, resulting in WF attacks with high classification accuracy. Based on CNNs, we built an SGAN model with a generator and a discriminator, in which the discriminator becomes a $K+1$ class WF classifier (K is the number of websites in the labeled set). This classifier utilizes three different sources of training data: labeled website traces collected by the attacker, unlabeled websites traces that could be from a publicly available WF database or fresh Tor traffic collected by running entry guards or middle nodes, and fake website traces produced by the generator.

The combination of different training sources enables GANDaLF to learn from a broader perspective, which leads to more precise WF classification only using a few labeled samples for training. In contrast, the learning capacity of supervised WF techniques is limited to the data distribution when using a small set of training samples, which leads to significantly weaker performance in the limited-data setting.

Since GANDaLF needs multiple data sources, the choice of unlabeled data impacts its classification performance. SGAN [99] constructed both labeled and unlabeled datasets from the same data distribution. In a WF attack, however, this would require the attacker to collect a very large unlabeled set to be aligned with the labeled data, which contradicts the goal of low-data training. Thus, to investigate the applicability of SGAN to low-data WF, we studied how different the unlabeled data distribution is from the labeled data distribution if we construct them from different datasets.

We explored three datasets – WF-I, and WF-S, and the MNIST computer vision

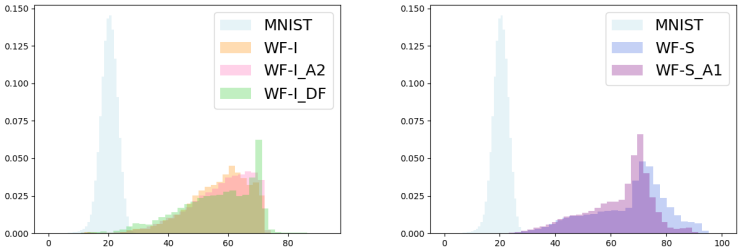


Figure 5.3: Distribution of euclidean distances between labeled and unlabeled data (**A1**: AWF [3] set consisting of 100 websites, **A2**: AWF set consisting of 100 websites (different from A2), and **D**: DF set [2]).

dataset to serve as a baseline. For MNIST, we built both labeled and unlabeled data from the MNIST set. For WF-I, we constructed the labeled set using the AWF1 set [3] and three different unlabeled sets: (i) one based on the same AWF1 set (WF-I in Figure 5.3), (ii) one based on the AWF2 set (WF-I-A2), and (iii) one based on the DF set collected with different network settings (WF-I-D). For WF-S, we used GDLF21 to be used as labeled set and build two unlabeled sets: (i) one from the same GDLF21 set (WF-S in Figure 5.3) and (ii) one from AWF1 (WF-S-A1). Then we computed the pair-wise Euclidean distances between labeled and unlabeled data in these settings; the distributions of these distances are shown in Figure 5.3.

Figure 5.3 shows that WF-I-A2 is close to WF-I, which indicates that if the traces are collected in the same network environment, their distance distributions are almost the same, even though they comprise different website traces. In WF-I-D, as the DF set was collected in different network settings, the distances had more variance. Similarly, the distance distributions of WF-S-A1 and WF-S became more different from each other, which we assumed as the most difficult data setup for GANDaLF. We will empirically show in Section 5.5 the impact of using unlabeled data chosen from other distributions, and conclude that it has minimal impact on the classification accuracy, but it does affect the stability of training and somewhat restricts the capacity of supervised learning.

With this preliminary analysis, we see that SGAN is promising to explore WF in the low-data setting by using a small labeled set together with a large unlabeled set to help train the discriminator. Furthermore, we will study more optimal labeled and unlabeled data settings to maximize the classification power of GANDaLF in Section 5.5.

Compared to MNIST with a normally distributed curve, however, Figure 5.3 shows more variance in the distances between labeled and unlabeled data for WF data. This means that we need careful tuning of SGAN to ensure better performance, which we discuss in Section 5.4.3.

In addition, we expect that WF-S is a more challenging task than WF-I, because traces in WF-S are more different than WF-I as they are plotted on a wider curve in Figure 5.3. This results in additional difficulty to simulate realistic fake subpage fingerprints as well as to classify fingerprints to correct website labels.

5.4.3 SGAN Optimization for GANDaLF

Saliman et al. [99] proposed several SGAN architectures optimized for different datasets, including MNIST, CIFAR-10, and SVHN. Among these architectures, we selected the one optimized for CIFAR-10 as our starting point, since it yielded better initial accuracy on the AWF1 set.

In this section, we discuss the technical challenges we addressed to find the optimal SGAN architecture for WF tasks and key design decisions. First, we found several aspects of SGAN to be problematic when applied directly to the WF problem.

- SGAN was built based on two-dimensional (2D) convolutional layers. As pointed out by Sirinam et al., however, network traffic features do not carry a meaningful 2D spatial pattern in the same way as the images that most CNNs operate on [2]. Thus, we had to incorporate one-dimensional (1D) convolutional layers into SGAN, and further, tune the model towards WF classification tasks. The application of 1D convolutional layers to SGAN revealed several additional problems that we needed to address to improve the performance of GANDaLF.

- SGAN used neither a batch normalization (BN) nor dropout in the generator. However, building the initial SGAN architecture to use 1D convolutional layers made training unstable. Thus, we explored whether adding BN or dropout layers to both generator and discriminator would help improve the training process.

- Saliman et al. proposed feature matching loss using the mean absolute difference (i.e., L1 loss) between the expected features of real data and the expected features

Table 5.2: Hyperparameter optimization showing the chosen parameters and search spaces for the WF-I and WF-S scenarios (**G**: generator, **D**: discriminator, [**Conv**]: 1D convolutional layer block, [**Full**]: fully-connected layer block, **Up**: Upsampling layer, **act**: activation function, and **#**: number).

Scenario →	WF-I		
HyperParam ↓	Choice		**Search Space**
	G	D	
[Conv] layer#	9	8	4∼12
[Conv] filter#	64∼ 512	32∼ 256	10∼1,000
[Conv] filter size	5	5	2∼10
[Conv] stride size	1	1∼2	1∼4
[Conv] dropout rate	-	0.3	0.2∼0.9
[Conv] act	ReLU	LeakyReLU	ReLU, LeakyReLU, ELU
[Conv] Up#	4	-	2∼9
[Full] layer#	1	5	1∼6
[Full] node#	316	512∼ 2,048	128∼2,048
[Full] dropout rate	-	0.5	0.2∼0.9
[Full] act	ReLU	ReLU	ReLU,LeakyReLU
input dim	5,000		5,000
z dim	100		50∼700
optimizer	Adam		Adam
learning rate	$2e^{-4}$	$5e^{-5}$	$1e^{-5} \sim 0.1$
epoch	≤30(CW), ≤150(OW)		10∼1,000
batch	32		16∼128

Scenario →	WF-S		
HyperParam ↓	Choice		**Search Space**
	G	D	
[Conv] layer#	9	8	4∼12
[Conv] filter#	32∼ 256	32∼ 256	10∼1,000
[Conv] filter size	20	20	2∼30
[Conv] stride size	1	1∼4	1∼4
[Conv] dropout rate	0.3	0.3	0.2∼0.9
[Conv] act	ReLU	LeakyReLU	ReLU, LeakyReLU, ELU
[Conv] Up#	4	-	2∼10
[Full] layer#	3	5	1∼5
[Full] node#	316	512∼ 2,048	10∼2,048
[Full] dropout rate	0.5	0.5	0.2∼0.9
[Full] act	ReLU	ReLU	ReLU, LeakyReLU
input dim	5,000		3,000∼8,000
z dim	100		50∼700
optimizer	Adam		Adam
learning rate	$2e^{-4}$	$5e^{-5}$	$1e^{-5} \sim 0.1$
epoch	≤10(CW), ≤100(OW)		10∼1,000
batch	16		16∼128

of the generated data. However, since webpage traces are different from image features, we also investigated different feature matching loss functions (L2 vs L1 distance).

- The choice of hyperparameters impacts the performance of SGAN. Thus, we had to empirically find the optimal parameters for WF-I and WF-S, respectively.

To overcome these limitations of the original SGAN architecture, we introduced the following technical innovations. Note that we used the same architecture for WF-I and WF-S, but we empirically selected hyperparameters for each scenario as shown in Table 5.2.

Deeper 1D-Based Design. The initial SGAN implementation [106] with feature matching was based on the generator containing four deconvolutional 2D layers [1] and a discriminator consisting of seven 2D convolutional layers. After simply switching from 2D convolutional layers to 1D layers, we trained it using 90 instances per website and it reached 78% CW accuracy in the WF-I setting. As shown in Figure 5.2, we added more 1D convolutional layers, which resulted in a higher accuracy. This change led to a classifier that obtained 95% accuracy with 90 training instances for each of the 100 websites.

Dropout and BN. We found that *selective* use of dropout layers and the full use of BN layers in the generator helps to make the training more stable in WF-S. More specifically, we added a dropout layer after all convolutional layers except the first and last layers as shown in Figure 5.2. In contrast, for WF-I, we only used BN layers in the generator, since the use of dropout layers in any location worsened the performance. Furthermore, we added several fully connected layers, followed by dropout layers between the flattened layers, where we captured features to compute the feature matching loss, and the last output layer.

Different Generator Loss. We noticed that the same L1 feature matching loss works properly for WF-I, while L2 loss improved the testing accuracy in the WF-S scenario more than L1 loss. However, in both scenarios, generator loss started with very low value

[1] A deconvolution is the inverse operation of the convolution, which means performing the convolution in the back propagation.

around 0 and did not decrease much, while discriminator loss continually decreased. This indicates that the generator did not generate actual good fake traces, while the supervised performance was constantly improved. This is because intra-class variation in WF traces is more significant than for images such as MNIST, which made it harder for GANDaLF to reduce the feature matching loss. Furthermore, when using AWF1 set as unlabeled data in WF-S, the generator loss kept increasing even as the discriminator loss was decreasing. We will investigate this problem in detail in Section 5.5.3.

Stride and Kernel Choice in WF-S. Furthermore, we found that a greater length of strides and kernels helped improve the performance of GANDaLF in WF-S. This was consistent with our expectation that increasing the stride length and kernel sizes, which shrinks the output volume after the convolutions, might lead the network to better handle WF-S having greater intra-class variation than WF-I and capture meaningful features. This resulted in the number of features used to compute the feature matching loss in WF-I being 20,224, while it was 1,280 in WF-S scenario. As such, losing some details by increasing the stride and kernel sizes helps to better capture the traffic pattern when features are more variable within each class.

Input Representation for WF-S. Most DL-based WF attacks represent a website trace as a sequence of ± 1's that indicate packet direction. In our investigations, we explored several alternative data representations, such as inter-packet delay (IPD) and Tik-Tok [97] sequences, for both WF-I and WF-S scenarios. In the WF-S scenario, we found that IPD yielded +9% and +8% better CW accuracy than the direction and Tik-Tok features. Hence, we used IPD sequences in WF-S scenarios throughout the chapter.

Parameter Tuning. Along with the architectural tuning, we also explored different combinations of parameters involved in the architecture for WF-I and WF-S. Since the GDLF dataset is different from the AWF dataset, we conducted hyperparameter tuning separately for each scenario. We used 90 instances of AWF1 and all of AWF2 for tuning WF-I, and 90 instances of GDLF25 and all of AWF1 for tuning WF-S. In this way, we can ensure that the overlap is minimal between the tuning sets and the testing sets used in Section 5.5.

The parameter search space and chosen parameters are reported in Table 5.2. Beyond these parameters, we also adjusted other components in SGAN. First, the SGAN of Saliman et al. [106] used weight normalization (WN) [109] in the discriminator, while we applied batch normalization since WN barely impacted the performance, and BN is easier to implement. Second, we applied different learning rates to the discriminator and generator during the optimization based on findings by Heusel et al. [110] that this ensures better convergence to Nash equilibrium, and further, led GANs such as DCGAN [107] to achieve better performance.

Summary. Overall, the most effective design for GANDaLF is to go much deeper by adding fully-connected layers and more convolutional layers. As shown by Sirinam et al. [2], more layers help the model learn the inner structure of website traces more effectively since WF set has more inter- and intra-class variances than the image set. On the downside, this may make the model more complicated, resulting in more chances of overfitting. Thus, we added dropout and batch normalization layers to relieve this concern.

5.5 Evaluation

In this section, we evaluate the performance of GANDaLF in various experimental scenarios. First, we compare GANDaLF to the state-of-the-art WF techniques in the WF-I setting (index pages) with limited training data. Then we further investigate the applicability of GANDaLF and other data-limited attacks in the WF-S setting (subpages).

5.5.1 Experimental Setting

Setup.

We implemented GANDaLF using Tensorflow; each experiment was conducted on a Tesla P100 GPU with 16GB of memory. Using pseudocode, we provide details of the GANDaLF experimental setup in Algorithm 1. We evaluated each technique using five trials and added more experiments up to a maximum of 20 when the standard deviation was greater than 1%.

Algorithm 1: GANDaLF training.

Input : Labeled examples $(x_l, y_l) \sim p_{d_1}$, Unlabeled examples $(x_u) \sim p_{d_2}$, latent variable $z \sim p(z)$, number of iterations i, α_1=$2e^{-4}$, α_2=$5e^{-5}$, and β=0.5.

G – generator network
D – discriminator network
f – output of the flatten layer of D
L_D – discriminator loss
L_G – generator loss
ω – parameters of discriminator
θ – parameters of generator
for *i = 1 to m* **do**
  $\tilde{x} \leftarrow G(z)$
  $\hat{x_z}, f_z \leftarrow D(\tilde{x})$
  $\hat{x_l}, f_l \leftarrow D(x_l)$
  $\hat{x_u}, f_u \leftarrow D(x_u)$
  $L_s \leftarrow \text{CrossEntropy}(\hat{x_l}, y_l)$
  $L_u \leftarrow$ - logsumexp$(\hat{x_u})$ + softplus(logsumexp$(\hat{x_u})$) + softplus(logsumexp$(\hat{x_z})$)
  $L_D \leftarrow L_s + L_u$ `/* supervised+unsupervised loss */`
  $\omega \leftarrow \text{Adam}(\nabla_\omega \frac{1}{m} \sum L_D^{(i)}, \omega, \alpha_2)$ `/* D optimizer */`
  $L_G \leftarrow \text{MAE}(f_z, f_u)$ `/* MSE in WF-S */`
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum L_G^{(i)}, \theta, \alpha_1, \beta)$ `/* G optimizer */`
end

To implement state-of-the-art WF techniques, we adopt the original implementations provided by researchers [82, 88, 2, 10, 11]. We made few changes when necessary for the data loading pipeline and for hyperparameter tuning. When tuning k-FP, we explored different numbers of trees from 500 to 2,000 and finally chose 2,000 for both scenarios. For DL-based WF attacks, we explored different mini-batch and convolutional stride sizes, as these are parameters that are significant for GANDaLF. Both DF and TF were also allowed to train for additional epochs until validation loss increased for five consecutive epochs. Since TF [10] used 1-20 training instances per website for the N-shot learning, we chose similar training set sizes, however, we increased the size up to 90 instances to see how much DL-based classifiers are benefited by additional training instances. That is, to construct the training labeled set, we randomly sampled 5, 10, 20, 50, and 90 instances for 100 websites in WF-I. In WF-S (subpages), we randomly chose one instance using each of n_s subpages per site in which n_s = 5, 10, 20, 50, and 90 (i.e., total $1 \times n_s$ instances).

For GANDaLF, we randomly sampled these instances rather than using one subpage per site, since this approach yielded slightly higher closed world accuracy, which will

be detailed in Section 5.5.3. For other classifiers, we chose $1 \times n_s$ achieving a higher accuracy. In either case, the standard deviations between trials in WF-S are greater than WF-I, most likely due to much larger intra-class variance.

Metrics.
We summarize the metrics for CW and OW evaluation as follows.

- *Accuracy*: The percentage of predictions that are correct. This metric is traditionally used to evaluate classifiers in the CW setting in prior WF work, which we adhere to.
- *Precision*: The percentage of positive predictions (i.e. predicted as "monitored") that are correct. If the classifier is tuned for high precision, it minimizes the number of users being misdetected as "guilty," but may miss some instances that were truly monitored.
- *Recall*: The percentage of monitored-site instances that are classified as "monitored." A classifier tuned for high recall will reliably identify when a sensitive site has been visited, but may also misidentify "innocent" websites as sensitive.

A WF adversary must consider both the precision and recall of their classifier when evaluating the results of a real-world attack, so we show precision-recall curves for our OW experiments.

5.5.2 Fingerprinting Websites with Index Pages

In this section, we evaluate the classification ability of GANDaLF and other WF techniques in a low-data setting by training and testing with website index pages.

CW Performance.
We trained GANDaLF, k-FP, DF, Var-CNN, and TF classifiers using 5-90 instances per website, randomly sampled from the AWF1 dataset. To train GANDaLF, we used AWF2 as the unlabeled data. For a fair comparison, we also used the AWF2 dataset for the pre-training phase of the TF attack. The performance for each technique is shown in Table 5.3. The best results for a given number of training instances is *shown in bold.*

Table 5.3: Comparison to k-FP, DF, Var-CNN, and TF using 5-90 training instances. We do not show standard deviations less than 1%. We measured the time (**s**: seconds) for testing 42k testing samples. Other numbers are %.

TrainN	GANDaLF	k-FP	DF	Var-CNN	TF
5	70±2	61	60±2	25.9	**78±1**
10	**81±1**	72.5	79±2	69.1	**81.6**
20	87±1	77.3	89±2	**90.8**	83.1
50	93±1	82.8	95.1	**97.1**	83.9
90	95±1	85.5	97.1	**98.3**	84.2
time	5.5s	1.1s	7.6s	43.6s	8.5s

Our experiments show that GANDaLF is effectively tied with TF when using 10 samples per class. However, the testing cost of GANDaLF was lower than TF, DF, and Var-CNN. For 50 samples and above, Var-CNN is the best classifier, but it was much less effective when limited to 5 or 10 samples, with accuracies of 26% and 69%, respectively. In the lowest data setting with 5 samples, TF was the most accurate classifier due to its pre-trained WF model. Across all classifiers, if the attacker can afford this larger cost for data collection, the payoff is worthwhile for closed-world classification of index pages. In particular, when either DF or Var-CNN is trained on many more instances, performance is much improved. When trained on 90 instances, the accuracy of DF and Var-CNN improves to 97% and 98% respectively. This is important evidence to suggest that these models require very large labeled training datasets to learn effective feature representations in the WF-I scenario.

Table 5.4: Impact of circuit diversity on labeled training data (DF set [2]). We used DF as labeled data and AWF2 as unlabeled data. All standard deviations are less than 0.5%.

Train (25)	Acc	Train (90)	Acc
1 circuit	86.6	slow	93.4
5 circuits	86.8	fast	92.9
40 circuits	87.1	random90	93.5

Impact of Circuit Diversity.

The attacker using GANDaLF needs to gather and use a smaller dataset of labeled data than in other attacks, so the source of that data may impact attack accuracy.

Algorithm 2: Data sampling to generate labeled sets by varying the circuit diversity.

Input : DF dataset ($D = (X, Y_l, Y_c)$), total circuit index array ($C = \{1, 2, ..., 40\}$), circuit count (n_c), website count (n_w), labeled sample count per class (n_l), and testing sample count per class (n_t).

Output: Training data (I_{tr}) and testing data (I_{te}).

Shuffle D. /* (samples, labels, circuit labels) */
Shuffle C.
Initialize C_{tr}, C_{te},D_{tr}, D_{te}, I_{tr}, I_{te}.
$C_{tr} \leftarrow$ randomly chosen n_c entries in C.
for *(x, y_l, y_c) in D* **do**
    if *y_c in C_{tr}* **then**
        $D_{tr} \leftarrow D_{tr} \cup (x, y_l)$
    end
    if $n_c < 36$ **then**
        /* To ensure that circuits in C_{te} should have at least 9,500 entries since each circuit subset consists of 95×25. */
        $C_{te} \leftarrow \{C - C_{tr}\}$.
        if *y_c in C_{te}* **then**
            $D_{te} \leftarrow D_{te} \cup (x, y_l)$
        end
    else
        $D_{te} \leftarrow$ randomly chosen $n_w \times n_t$ entries in $\{D - D_{tr}\}$
    end
end
/* sampling for each website subset. */
for *i in $(\{1, 2, ..., n_w\})$* **do**
    $I_{tr} \leftarrow$ randomly chosen n_l instances in D^i_{tr}
    $I_{te} \leftarrow$ randomly chosen n_t instances in D^i_{te}
end

Algorithm 3: Data sampling to simulate the victims with fast or slow circuits.

Input : DF dataset ($D = (X, Y_l, Y_c)$), total circuit index array ($C = \{(1, t_1), ..., (40, t_{40})\}$), website count ($n_w$), labeled sample count per class (n_l), and testing sample count per class (n_t).

Output: Training data (I_{tr}) and testing data (I_{te}).

Shuffle D. /* (samples, labels, circuit labels) */
Shuffle C. /* (circuit index, mean of site loading time). */
$C_{index} \leftarrow \{1, 2, ..., 40\}$ /* index. */
$C_{time} \leftarrow \{t_1, t_2, ..., t_{40}\}$ /* mean loading time. */
Initialize C_f with top 4 indices in reverse(C_{time}).
Initialize C_s with top 4 indices in C_{time}.
Initialize C_{tr}, C_{te},D_{tr}, D_{te}, I_{tr}, I_{te}.
if *choice == "fast"* **then**
    $C_{te} \leftarrow C_f$ /* use fast subsets as testing data. */
else
    $C_{te} \leftarrow C_s$ /* use slow subsets as testing data. */
end
$C_{tr} \leftarrow \{C_{index} - C_{te}\}$
for (x, y_l, y_c) *in* D **do**
    if y_c *in* C_{tr} **then**
        $D_{tr} \leftarrow D_{tr} \cup (x, y_l)$
    end
    if y_c *in* C_{te} **then**
        $D_{te} \leftarrow D_{te} \cup (x, y_l)$
    end
end
/* sampling for each website subset. */
for i *in* $(\{1, 2, ..., n_w\})$ **do**
    $I_{tr} \leftarrow$ randomly chosen n_l instances in D^i_{tr}
    $I_{te} \leftarrow$ randomly chosen n_t instances in D^i_{te}
end

In particular, the circuits used to collect this data might be slow, fast, or otherwise not representative of the kinds of conditions faced by the victim. To investigate the impact on GANDaLF of the diversity of circuits used to gather labeled training data, we examine how the number of circuits used to gather data impacts accuracy. We used a subset of the DF dataset collected using 40 circuits, and split it into 40 smaller subsets, one per circuit. Each subset consists of 25 instances of each of 95 websites. Thus, we randomly sampled 95 websites and 25 instances (95×25) within one subset, four subsets, and 40 subsets to construct three training labeled sets and 100 instances of each of 95 websites within all 40 subsets to build one testing set (We detailed this data sampling in Algorithm 2). Then we trained three different models using each labeled set and tested them using the testing set. As shown in Table 5.4, the performance somewhat improved with increasing number of circuits, though it is far from critical in performing the attack.

Impact of Network Conditions.
While the attacker would likely use multiple circuits to gather labeled training data, the victim may have a particularly slow or fast circuit. Thus, we examine how the speed of the victim's circuit impacts the attack. We use the same 40 subsets of the DF dataset as when testing circuit diversity. We split out the four fastest circuits and the four slowest circuits by using the total website load times.

We then constructed fast (or slow) testing sets by randomly sampling 95×100 instances from data gathered using the four fast circuits (or slow circuits), which was the same testing set size used in DF [2]. To train GANDaLF, we randomly chose 95×90 instances over the remaining 36 subsets. We described this data sampling details in Algorithm 3.

As a baseline, we further trained GANDaLF using randomly chosen 95×90 samples over 40 subsets and tested it using another randomly chosen 95×100 instances over 40 subsets. As shown in Table 5.4, GANDaLF performs modestly worse when identifying traces using fast circuits and about the same on slow circuits. The small margins indicate that the network condition when collecting the victim's traffic minimally impacts the performance of GANDaLF.

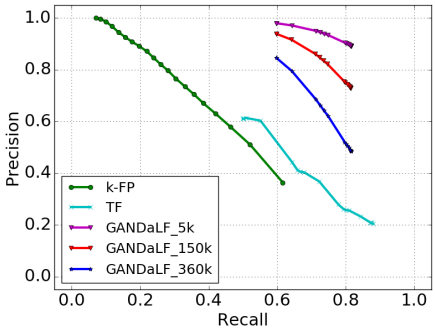


Figure 5.4: Comparison to k-FP and TF. We used 360k background traces for k-FP and TF.

Impact of Unlabeled Data.

To understand the impact of the choice of unlabeled data, we also trained GANDaLF using the AWF1 set as both labeled and unlabeled data. In other words, this models the case that both sample groups come from the same distribution. Interestingly, this change only led to a 1% increase in the accuracy of CW classification. We further trained GANDaLF using the DF set (which was collected in different network settings) and GANDaLF yielded 87% CW accuracy when using 20 labeled training instances per website. Even though the distributions of distances between labeled and unlabeled sets were somewhat different as shown in Figure 5.3, this result shows that the gap did not critically impact the classification ability of GANDaLF. This suggests that the unlabeled data does not require either any of the monitored sites in the labeled set or the same network setting for the unlabeled data collection to provide a useful basis for semi-supervised learning.

OW Performance.

Since GANDaLF and TF performed more effectively in the low-data CW setting (i.e., 5-10 instances), we further evaluated them in the open-world scenario. In this evaluation, the classifiers were trained using 20 instances for each monitored site in AWF1 and 2,000 unmonitored site instances from AWF-OW. We then tested using a background set of 360,000 unmonitored website samples, which is the same size as the largest background set explored by TF in their OW evaluations [10]. We further varied the size of the unmonitored set from 5,000 to 360,000 to show the impact of the background set on the

performance of GANDaLF. As shown in Figure 5.4, GANDaLF outperformed k-FP and TF and increasing the unmonitored set size degraded GANDaLF performance. Compared to the CW setting, GANDaLF provides better performance than TF by a more significant margin in detecting monitored websites versus unmonitored websites. The better effectiveness in OW scenarios is mainly because the discriminator using additional supervised loss also became more benefited by the binary classification setting.

5.5.3 Fingerprinting Websites with Subpages

In this section, we investigate the classification ability of GANDaLF and other techniques in the WF-S setting. This scenario not only represents a more realistic scenario for attacks, but also a more challenging one, as the inclusion of many subpage traffic instances results in high intra-class variation.

Table 5.5: Comparison to k-FP, DF, Var-CNN (Var), and TF using 5-90 training instances. For unlabeled sets, we used AWF1 (**GF(A)**) or GDLF-OW-old (**GF(G)**). We do not show standard deviations less than 1%. We measured the time (**s**: seconds) for testing 12k testing samples. Other numbers are %.

TrainN	GF(A)	GF(G)	k-FP	DF	Var	TF
5	30±1	31±2	**41**	4	5±1	14±1
10	39±3	38±2	**46**	5±1	6±2	17±1
20	46±3	47±3	**52**	47±2	9±2	18
50	**57±2**	**56±3**	**57**	49±2	21±6	18
90	**62±1**	**62±3**	61	61	25±7	19
time	2.3s	2.3s	3.1s	5.1s	12.07s	8.2s

CW Performance.

For the WF-S CW experiments, we trained each technique in a low-data setting with between 5-90 training instances per site, where each instance was randomly sampled. This means that, at best, the attacker is able to train on one sample per subpage within each site in our dataset. Consequently, during experiments where the training sample count is below 90, there are subpages within the testing set on which the attack did not train. We believe this challenging CW scenario appropriately models the real-world difficulty of accurately profiling an entire website under reasonable data restrictions.

As shown in Table 5.5, this difficult training scenario and the higher intra-class

Table 5.6: GANDaLF CW accuracy (**Acc**) according to different labeled set by varying the number of subpages (s) and instances (i) in the labeled set. All numbers are %.

$s \times i$	2×10	10×2	20×1	random
Acc	42.78±2.3	46.03±3.5	45.91±2.0	46.7±2.0

variance reduced the performance for all WF methods. GANDaLF performed the most effectively using 90 instances per site and ties with k-FP when using 50 instances per site. In the lower data settings, however, k-FP is more accurate. We believe that the *categorical* features such as the total number of packets enabled k-FP to gain an enhanced understanding about subpage traces even using more limited training data. In contrast, deep learning models must learn feature representations from scratch using the few training samples provided, inevitably resulting in the network gaining a poorer understanding of the data.

TF and Var-CNN achieved worse performance than anticipated for all cases. The poor performance of Var-CNN may be explained by how heavily tuned the model is to the traditional WF-I scenario. The expanded receptive field of the dilated convolutions used by the network may cause the model to miss meaningful local patterns.

For TF, it seems that the distinctions between AWF websites did not help the model generate good features for the subpage traces, because the decision boundary for the classification in WF-S was different from WF-I. Since the pre-trained model was trained using labels, the decision boundary became more biased towards WF-I, leading to poor feature embeddings for subpage traces.

In contrast, GANDaLF was trained by additional unsupervised loss and feature matching loss, enabling it to learn a broader view of AWF1 traces without labels rather than focusing on the differentiation between AWF websites based on labels. This makes GANDaLF performance solid even when learning from a different distribution (WF-S versus WF-S-A1 in Figure 5.3).

Impact of Subpages in Labeled Data.

We further varied the number of subpages used to represent each website in the training data. More specifically, we varied the number of subpage classes by fixing the total instance count at 20. For example, we varied the subpage count s and instance count i to be $s \times i = 20$. We first randomly selected s subpages among 96 subpages and

then randomly sampled i instances for each subpage. This scenario allows us to better see the effects of the increased intra-class variance as the number of subpages (i.e., s) increases to labeled training data and further guides us to build the optimal labeled set to maximize the GANDaLF performance.

Based on Table 5.6, building the labeled set by random sampling without considering s performed slightly better than other cases while GANDaLF performance remarkably worsened with more limited subpages when s=2. This indicates that enough variance between subpages in the labeled set is important to maximize the performance of GANDaLF.

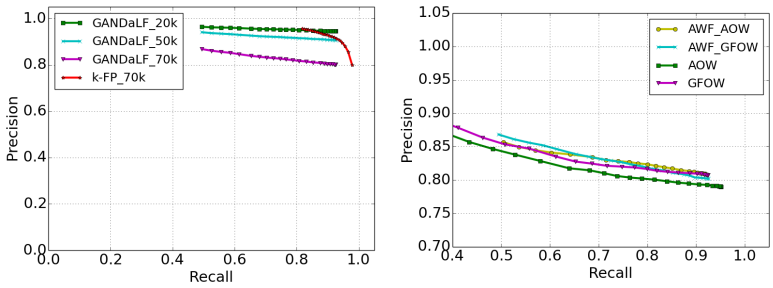
Impact of Unlabeled Data.

As briefly discussed in Section 5.4.3, we studied the effect of unlabeled sets to the classification performance as well as the generator loss. In this experiment, we used two unlabeled sets, AWF1 and GDLF-OW-old which has a three-month time gap with GDLF25 set, and one labeled set from GDLF25. The generator loss somewhat decreased with GDLF-OW-old while it kept increasing with AWF1. Even though the GDLF-OW-old set made training more stable, the CW accuracy was comparable across most settings based on Table 5.5 (GF(A) versus GF(G)). However, the use of unlabeled set built from different data distribution degraded the generator ability, which led to more biased training towards a better discriminator and may eventually result in limiting the upper-bound of supervised learning capacity since L_u in Algorithm 1 also hardly decreased.

To create good fake samples against a greater number of classes than examined by Saliman et al. [99], we should feed a much larger unlabeled set of subpage traces in which the corpus of the websites, subpages, and instances is tremendous to train the generator effectively. As a result, both generator and discriminator may reach the optimal Nash equilibrium while gaining powerful supervised performance with a high CW accuracy. We leave further investigation on the usage of more optimal unlabeled data as future work.

OW Performance.

We further conducted an OW evaluation of GANDaLF and k-FP in the WF-S setting, since they had better performance than other WF attacks in the CW evaluation. For this



(a) Comparison to k-FP. (b) Impact of unlabeled set.

Figure 5.5: GANDaLF OW experiment by varying the background sizes and unlabeled sets.

Table 5.7: Various unlabeled data settings using AWF1, AWF-OW (**AOW**), and GDLF-OW-old (**GOW**). We reported the trace count (**size**), whether or not the network setting was different from the GDLF25 setting (**network**), and time gap (**y**: years, and **m**: months).

setup	AWF1-AOW	AWF1-GOW	AOW	GOW
size	649k	329k	400k	80k
network	no-no	no-yes	no	yes
timegap	3y-3y	3y-3m	3y	3m

experiment, we trained both classifiers using the labeled set consisting of 90 instances of 25 monitored sites and 2,250 unmonitored subpages. In addition, we used AWF1 and AWF-OW sets as unlabeled data for GANDaLF. Figure 5.5a shows that as we increase the size of the unmonitored set, GANDaLF becomes less effective, as expected. In particular, k-FP outperformed GANDaLF in the OW setting, which indicates that the handcrafted features provide a more consistent basis to identify pages from sites in the monitored set than the GANDaLF model.

We further investigated how combining unlabeled sets could amplify the performance of GANDaLF. For this experiment, we adopted AWF1, AWF-OW, and GDLF-OW-old. We created combined datasets of AWF1 with AWF-OW and AWF1 with GDLF-OW-old, and compared these against AWF-OW and GDLF-OW-old by themselves. See Table 5.7 for details.

Figure 5.5b shows that both of the combined unlabeled sets performed slightly more effectively. This suggests that the amount and perhaps variety of unlabeled data played

a role in enhancing the performance of GANDaLF, even though some of the data was collected three years prior to the labeled data and from different network conditions (i.e., AWF-AOW in Table 5.7). Furthermore, GANDaLF improved with the inclusion of the GDLF-OW-old set, which suggests that the unlabeled subpage traces help generate good fake samples to distinguish monitored subpages from unmonitored subpages by lowering L_u and L_G in Algorithm 1.

Summary.

We find that GANDaLF outperforms other DL-based classifiers on subpages. Surprisingly, however, k-FP was even more effective in both the CW and OW settings. The greater intra-class variation made it harder for automatic feature extraction to work effectively, while manually defined features can still work consistently in such a challenging setting.

Chapter 6

More Efficient Correlated Flow Traffic Analysis

The Tor design [111] explicitly acknowledges that such attacks can be effective and concentrates on a more limited threat model. In turn, many published analyses of the security of Tor treat flow correlation as a core primitive and simply account for the fraction of flows that can be observed by an adversary [30, 31, 32, 26, 6, 33, 29, 38, 39, 36, 37, 40]. These works typically describe methods to increase or limit the fraction of flows that an adversary can observe through some combination of internal manipulation of Tor protocols, manipulation of the Internet routing infrastructure, or network positioning and resources, and assume that flow correlation will work on these observations.

Despite this assumption, the extent to which end-to-end flow correlation attacks are a realistic threat against the Tor network remains somewhat unclear. One problem with the direct application of these attacks to Tor traffic is that traffic between the client and entry relay is not identical to traffic between the exit relay and the destination server, due to a variety of factors such as multiplexing of encrypted traffic; the use of fixed-size cells to carry data between Tor nodes; and delays caused by buffering, congestion, interaction between circuits, and Tor's flow control mechanisms. For example, when Sun *et al.* [6] applied Spearman's rank correlation to a small set of Tor flows, they found that nearly 100MB of traffic was required per flow to obtain adequate performance.

Nasr, Bahramali and Houmansadr [5] addressed this limitation by using deep neural networks (DNNs) to learn a more effective Tor-specific flow correlation metric; the resulting metric, DeepCorr, classifies pairs of flows as correlated or uncorrelated with very high accuracy using much less traffic.

However, another fundamental limitation of existing work on end-to-end correlation of Tor flows is the *pairwise* nature of the attack. To decide if a flow entering and leaving the Tor network are on the same Tor connection, these attacks compute the correlation between the two flow vectors (consisting of packet times and sizes); to deanonymize a set of flows, the attacks must compute the correlation between all possible incoming and outgoing flows. Thus, to deanonymize N Tor connections, they will perform N^2 comparisons.

This approach results in poor Bayesian Detection Rates (BDRs) since it has a very low *base rate*, the probability that both ends of a Tor connection are correlated (i.e., $\frac{1}{N}$). In particular, any correlation metric with a fixed False Positive Rate (FPR) of ρ will incorrectly classify ρN exit flows as being correlated with each entry flow, so that the probability that a given pair is actually correlated given that the metric classifies them that way is $\frac{1}{1+\rho N}$, which approaches 0 as N increases. Concretely, DeepCorr was shown to achieve a FPR of around 10^{-5} when tuned to minimize False Positives (FPs); so when trying to correlate a particular exit flow with the roughly 10^6 simultaneous entry flows, it would incorrectly label 10 entry flows as correlated, so that even if the True Positive Rate (TPR) was 1, the BDR would be only 9%.

This chapter presents a novel approach to flow correlation designed to further reduce the number of FPs between pairs of flows to improve the BDR of this pair-wise comparison. Our approach uses a novel DNN architecture and training strategy to extract features of highly-correlated flow pairs. Then, we divide flows into multiple short time segments (which we call *windows*) by allowing some overlap between windows and correlate flow pairs in each window to *amplify* the difference in True Positives (TPs) and FPs. We call the resulting attack DeepCoFFEA, for Deep Correlated Flow Feature Extraction and Amplification.

At a high level, DeepCoFFEA works by learning two DNN models, G and H, that can be applied to entry flows (called *Tor flows* since they are wrapped in the Tor cell protocol) and exit flows, respectively; G and H should have the property that if Tor

flow t and exit flow x are correlated, then $d(G(t), H(x)) \geq \tau$ for some correlation metric d and thresholds τ, but if they are not correlated, then $d(G(t), H(x)) < \tau$. Then, we apply G or H to k consecutive flow segments (windows) to extract *feature embedding* vectors such that a larger fraction of correlated pairs of flows (than non-correlated pairs) will have a strong correlation measured by d. If two flows are seen as correlated by d in at least $k - \ell$ windows, we say the flows are correlated, and otherwise they are not; this exponentially amplifies the difference in TPs and FPs.

Note that DeepCoFFEA is still *efficient* despite its pairwise computation of correlation metrics since we only compute *lower*-dimensional $2kN$ feature embedding vectors (i.e., N Tor and N exit vectors) rather than computing $N \times N$ DNN. Furthermore, our evaluation is based on a cosine similarity matrix requiring relatively lower complexity than previous metrics. We highlight that these results are at a much lower computational cost than DeepCorr.

In practice, G and H may not always produce matching output on correlated windows. This is addressed by "aggregating votes" across multiple windows. We show that DNNs can be used to develop the feature extractors G and H, resulting in significantly higher TPRs with higher BDRs than DeepCorr on the same data set (97% TPR vs. 6% TPR to yield 99% BDR). We further demonstrate that DeepCoFFEA (1) successfully identifies correlated flows on disjoint evaluation and training circuits; (2) can effectively identify flows collected several months later than training data; and (3) can identify correlated flows protected with the obfs4-iat1 padding protocol with 61% TPR and 10^{-1}% FPR. In particular, DeepCoFFEA is capable of performing the correlation investigation for both defended and undefended traces at the same time by training on both types of flows.

Combined with the DeepCorr results and recent advances in website fingerprinting [2], these results show an urgent need to develop and deploy traffic analysis countermeasures to protect the users of Tor.

We summarize our key contributions as follows.

- **Correlated Flow Feature Extraction: A Novel Attack Framework.** We introduce a new attack framework by training DNN models using windows, testing the models against flows in each window, and aggregating the voting results of

multiple windows. We show that this strategy improves the resulting predictions, significantly extending the state-of-the-art approach at a reasonable computational cost. In particular, DeepCoFFEA can conduct the correlation analysis of 10,000 connections 540 times faster than DeepCorr. Furthermore, by creating overlapping windows, we can further boost the amplification capability. This improved strategy enables DeepCoFFEA to gain much higher TPRs with 94% TPR and 0% (empirical) FPR (i.e., 100% BDR).

- **New Architecture and Training Methods.** We develop feature embedding networks to learn two types of embeddings, namely Tor and exit flow embeddings, by extending Triplet Fingerprinting (TF) [10] and FaceNet [112] to be more suited to Correlated Flow Feature Extraction on Tor. This chapter also includes an empirical study to find the optimal DeepCoFFEA implementation, network architecture, features, and similarity metric (d) to achieve the best performance of DeepCoFFEA. We further evaluate DeepCoFFEA in various experimental settings using flows collected with arbitrary circuits, defense protocol, and some time gap to training flows, to show the viability of DeepCoFFEA-style attack in the real world.

- **Focus on BDR.** DeepCoFFEA is the first flow correlation to be evaluated using BDR. As Juarez *et al.* [52] pointed out, a low base rate can reduce the success rate of traffic analysis attacks even when there are high TPRs and low FPRs. This could be seen as one reason for skepticism about the threat of state-of-the-art flow correlation attacks at scale. However, we show that due to the use of amplification, DeepCoFFEA can achieve high BDR with much lower FPs, even at scale. We evaluate the BDRs of state-of-the-art flow correlation attacks and DeepCoFFEA, showing that, as expected, DeepCoFFEA significantly outperforms prior work.

6.1 Motivation

In this section, we sketch the adversary models of DeepCoFFEA and then discuss why and how DeepCoFFEA improves the state-of-the-art correlation metrics.

Threat Model.
As shown in Figure 2.2, we assume a passive network adversary who is able to monitor "Tor" traces $t_0, t_1, \ldots$ between clients and tor guards, and "exit" traces $x_0, x_1, \ldots$ between exit relays and destination servers, but does not interfere with these traces. After observing a pair of such flows t and x, the adversary conducts a correlation analysis between t and x to identify whether the two flows are on the same Tor connection, which we then refer to as "correlated" or "uncorrelated". This analysis results in leaking the identities of clients and servers, which breaks the anonymity of Tor. The adversary builds input feature vectors based on packet timing and size and identifies whether t is correlated to x using machine learning techniques or other correlation techniques, such as cosine similarity (cos) [26]. Adversaries can perform this analysis by running their own relays or controlling autonomous systems (ASes).

Problems in State-of-the-art Attacks.
In previous work on flow correlation attacks [6, 5], the adversary had a correlation metric $d(t, x)$ and computed $d(t_i, x_j)$ for every pair of traces observed in a given time window and if $d(t_i, x_j) \geq \tau$ for some threshold τ, concluded that t_i and x_j were "the same" flow. This approach requires adversaries to collect long flow sequences in the case of RAPTOR [6] or to evaluate an expensive metric on all n^2 flow pairs of n connections to use the DNN classifier [5].

For example, in evaluating the RAPTOR attack [6], Sun *et al.* computed Spearman's rank correlation coefficients for every pair of 50 Tor connections, where each connection captured *300 seconds* of traffic, and selected the exit trace with the highest coefficient as the best match for the Tor trace. DeepCorr [5] trained CNN models to learn a metric $d(t, x)$ and then, computed this metric using all pairings of 5,000 input and output flows, where the number of associated pairs was 5,000 and non-associated pairs was $5,000 \times 4,999$, in the testing phase. In particular, they built feature vectors based on two-dimensional arrays, $F_{i,j}$=[t_i;x_j], where $i, j \in \{1, ..., 5,000\}$ and trained models to

minimize the following loss function:

$$-\frac{1}{|D|}\sum_{F_{i,j}\in D} y_{i,j} log\Psi(F_{i,j}) + (1-y_{i,j}) log(1-\Psi(F_{i,j}))$$

where D is the training set, $y_{i,j} = 1$ if t_i and x_j are correlated and $y_{i,j} = 0$ otherwise, and Ψ is the output logit of the DeepCorr network.

All previous passive correlation attacks were based on a pairwise flow comparison, which can be problematic because even if the metric M is good, it will yield an impractically low base rate leading to a poor BDR, because each t_i is compared to every x_j. For instance, even if the FPR is 10^{-5} and there are a million flows entering the Tor network, on average, each x_i will be matched with 10 uncorrelated Tor flows, giving a BDR of 0.1, at most.

Our Approach.

We introduce a new type of correlation attack, which addresses both the computational complexity and the low BDRs of previous approaches to flow correlation in Tor using a combination of two techniques: Feature Embedding Networks (FENs) and Amplification.

To avoid directly learning based on all pairs of flows, DeepCoFFEA first trains a DNN consisting of three networks: *anchor*, *positive*, and *negative*, which are jointly trained using the triplet loss function. The anchor (a) model is evaluated on Tor traces while the positive (p) and negative (n) models are evaluated on correlated and uncorrelated exit flows, respectively, and share their weights. If we have two flow pairs, (t_1, x_1) and (t_2, x_2), the possible a, p, and n triplets, (I_a, I_p, I_n), will be (t_1, x_1, x_2) or (t_2, x_2, x_1). Using the triplet loss, DeepCoFFEA trains these embedding networks towards maximizing the similarity between a and p while minimizing the similarity between a and n.

This new type of network model makes the evaluation phase much less expensive because it only requires FENs to generate $2n$ lower-dimensional feature embeddings and then computes the pair-wise cosine similarity of n^2 embeddings. This substantially reduces the complexity of DeepCorr style attacks evaluating n^2 pairs to investigate n Tor connections. Furthermore, our models do not learn based on labels; rather, they learn statistical differences between correlated and uncorrelated flow pairs, leading to

more effective feature extraction as well as a generalized model to handle other types of testing sets. For example, by including some amount of defended flows in training data, FENs are able to detect both correlated defended and undefended traces simultaneously.

To reduce the number of FPs, we borrow the concept of amplification from randomized algorithms, in which a randomized decision procedure that has any significant gap between acceptance probabilities for positive and negative cases can be repeated multiple times to create a decision procedure with exponentially small false positive and false negative rates. In the context of DeepCoFFEA, we apply amplification by using "window partitioning" to divide the flow into several smaller non-overlapping or overlapping subflows (windows). Then, we evaluate each window separately and aggregate the results using voting in an ensemble fashion.

In particular, we train FENs to learn two functions, $G(t)$ and $H(x)$. These functions have the property that when t and x are correlated, $cos(G(t), H(x)) \geq \tau$ and when t and x are not correlated, then usually $cos(G(t), H(x)) < \tau$. By dividing t and x into k discrete windows and computing the similarity between G and H on the subflows in each window, we end up with k-dimensional vectors comprised of 1s if the subflows are correlated and 0s otherwise, which are *votes* from k windows. Then, we aggregate these votes to determine the final decision: if at least $k - \ell$ votes are positive, we say the flows are correlated, and otherwise we say they are uncorrelated. For example, if $k = 5$ and $\ell = 1$ and the vote is [1, 1, 1, 0, 1], for a given flow pair, this pair is predicted as *correlated*. By adjusting k and ℓ for a given anticipated flow set size n, we can push the false positive rate below the base rate of $1/n$, giving a BDR that does not trend asymptotically to 0.

Compared to DeepCorr [5], instead of learning a comparison metric $d(t, x)$, we learn a *pair* of functions $G(t)$ and $H(x)$ using the triplet loss. Then, the adversary computes $G(t)$ for every Tor trace and $H(x)$ for every exit trace in k successive windows. We expect only one exit trace, x_j, to line up with the same Tor trace, t_i, with at least $k - \ell$ 1 votes. We show that amplification can further reduce the number of FPs against a pair-wise cosine similarity computation and consequently, DeepCoFFEA becomes more effective with a much lower FPR and higher BDR than the state-of-the-art correlation techniques in Section 6.4.

6.2 DeepCoFFEA Attacks

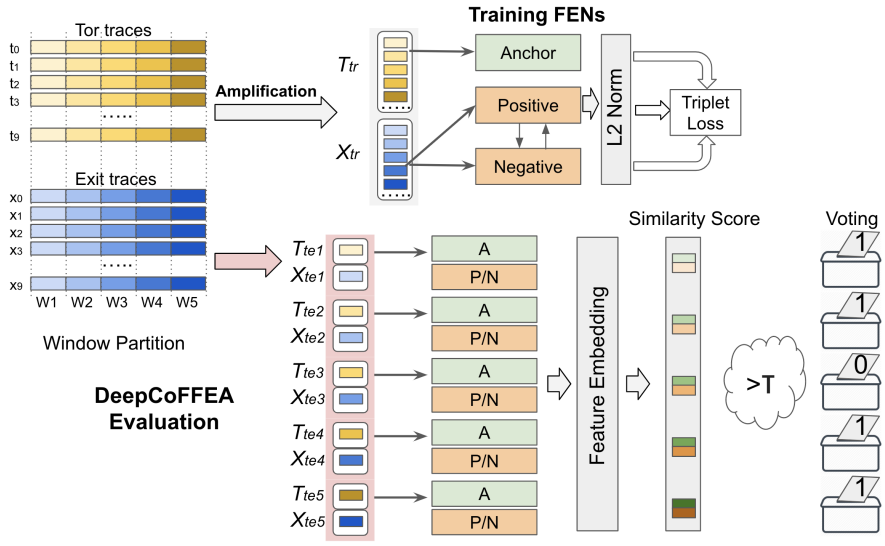


Figure 6.1: Example DeepCoFFEA Scenario: In this example, we had ten (t_i, x_i) flow pairs and five windows (W1,...,W5). First, we performed the non-overlapping window partition to generate two training sets, T_{tr} and X_{tr}, and ten testing sets, T_{te1},..,T_{te5},X_{te1},..X_{te5}. Then, we trained the DeepCoFFEA feature embedding network (FEN) with T_{tr} and X_{tr} and generated the feature embedding vectors using A and P/N models for each testing set, (T_{te_w},X_{te_w}) where w=1,...,5. We then computed the pairwise cosine similarity scores for each testing window and voted with 1 if the score was greater than τ or 0 otherwise. Finally, we aggregated those results and determined that the flow pair was correlated if it had at least four 1 votes.

In this section, first, we detail how we extended previous work [10, 112] to train the DeepCoFFEA FENs to generate Tor and exit flow embeddings. Next, we describe several methods to compute whether two embedded vectors are correlated and discuss the architecture of the DeepCoFFEA FENs and the hyperparameters involved. Finally, we describe our evaluation methodology with an example scenario.

6.2.1 Feature Embedding Networks for Correlation Study

To develop FENs for use in the DeepCoFFEA attack, we started with the TF network architecture [10] and adapted it for the flow correlation attack model. As a preliminary step, we tried using their networks directly (that is, having the A,P, and N networks

all share weights) when trained with Tor flow, exit flow, and exit flow triplets, but as expected we found that triplet loss did not decrease with training. We then made the following key changes to improve this initial result.

Two Different Networks.

Using the TF architecture, the triplet loss did not converge, due to two factors. First, Tor and exit flows are *different* traffic collected at different points in that the Tor trace is collected between the client and guard node and the exit traffic is collected between the exit relay and the web server. Second, if we used the same network for Tor and exit flows, it would increase the amount of padding to exit traces because they contained a relatively lower number of packets than the Tor traces. Thus, in contrast to FaceNet [112] and TF, we adopted two separate network models: one for the A network and another common model to the P and N networks. This approach led to reduction of the initial loss value, and further, helped achieve decreasing loss curves with training. However, we still ended up with an overall small drop in the training loss. For further improvement, we modified the triplet generator, as described next.

Triplet Epoch Generator.

The TF implementation chose triplets from positive and negative examples without regard to whether a particular negative had already been used in a previous input, which led to many exit flows being selected both as a positive (I_p) and as a negative (I_n). This quickly froze the triplet loss at some value because some flow pairs were used interchangeably to both maximize and minimize the correlation in the triplet loss function. To resolve this issue, we divided the exit flows into two sets and implemented the triplet generator to choose I_p from one set and I_n from the other set. In this way, we guaranteed that I_p and I_n were always different within a batch. However, we found that we could obtain a better loss curve when that guarantee was extended to an epoch. Note that we set 128 batches for an epoch in all experiments in the chapter. Thus, we kept shuffling the exit trace set and dividing it into two separate pools for every epoch. With this epoch generator, we were able to reach a training loss value closer to zero.

Loss Function.

We had to decide a correlation metric to be computed in the triplet loss function so we explored both cosine similarity and Euclidean distance metrics, as proposed by previous

research [112]. Interestingly, with the Euclidean distance function, the loss never decreased. Thus, the DeepCoFFEA FENs were trained to minimize the following triplet loss function:

$$\mathbf{L_{DeepCoFFEA}} = max(0, cos(G(I_a), H(I_n)) - cos(G(I_a), H(I_p)) + \alpha)$$

In other words, the goal of FENs is to learn embeddings G and H that satisfy $cos(G(I_a), H(I_n)) < cos(G(I_a), H(I_p)) - \alpha$. Further, to make the process of selecting hard-negatives more efficient, this margin α was also used as a boundary radius to select semi-hard-negatives. Thus, semi-hard-negatives are further from positives while being within distance α of the anchor point. As such, by tuning α, we could adjust the margin that was enforced between positive and negative pairs. We chose this value using hyper-parameter tuning in Section 6.3.3.

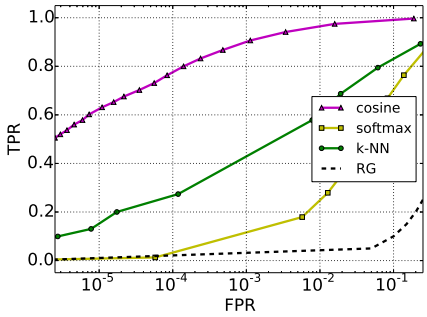


Figure 6.2: ROC with different evaluation methods (Note that x-axis is log scale and RG is Random Guess.)

6.2.2 Correlation Methodology

The DeepCoFFEA attack proceeds as follows. First, once the two FENs G (A network) and H (P/N network), are trained, we extract the Tor and exit flow embeddings, $G(t_{i,w})$ and $H(x_{i,w})$ in which t_i are the Tor flows, and x_i are exit flows, and $0 \leq i < n$, and $1 \leq w \leq k$ for n flow pairs and k windows. Second, we compute correlation metrics, $d(G(t_{i,w}), H(x_{j,w}))$ in which $0 \leq i, j < n$, for each window $w \in 1, \ldots, k$, for each of the n^2 potential flow pairings. Third, based on the thresholds, τ, we record a 1 vote if $d(G(t_{i,w}), H(x_{j,w})) \geq \tau$ and 0 otherwise. Finally, we decide that t and x are correlated

if they received at least $k - \ell$ 1 votes.

In this section, we empirically studied the cosine similarity, softmax function, and k-NN classifiers to compute d and the correlation thresholds τ.

Cosine Similarity.

The cosine similarity measures the cosine of the angle between two vectors projected in a multi-dimensional space. It captures the angle, not magnitude, such as the Euclidean distance. Since the FENs were trained using triplet loss based on the cosine similarity (cos), we naturally studied this similarity score as the similarity metric for DeepCoFFEA. That is, we computed the similarity scores for each window w of all Tor and exit embedding pairs, $(t_{i,w}, x_{j,w})$, in which $0 \leq i, j < n$ for n testing flow pairs. For each pair, if $cos(G(t_{i,w}), H(x_{j,w})) \geq \tau$ for some threshold τ, we recorded a vote of 1 and 0 otherwise. We present how we chose τ in Section 6.3.5.

Softmax.

We applied the softmax function, which normalizes the embedding into a vector following a probability distribution with a sum of 1, to the feature embeddings. Based on this probability vector, we could determine a predicted "label" by taking the logit with the highest probability. To investigate the Tor and exit flow pair (t, x), we computed the top-h labels for t and x based on argsort(soft($G(t)$)) and argsort(soft($H(x)$)) in which soft(x_i)=$\frac{e^{x_i}}{\Sigma_{j=1}^{n} e^{x_j}}$. We chose h=10 as it led to a relatively better TPR. If the top 10 labels had at least one common label, each window was voted with 1, and 0 otherwise.

k-NN with Clustering.

We trained k-NN classifiers using the Tor flows and tested them using the exit flows. We further trained the classifiers using the exit flows and tested them using the Tor flows to find the best setting. For either direction, we first had to label the flows in the training set before training. We explored k-means [113] and Spectral clustering [114] to label training flows. For both clusters, we varied k from 5 to 295, resulting in 5-295 labels to be trained. After training k-NN models using Tor (or exit) flows labeled by clustering, we evaluated the models using exit (or Tor) flows. We then conducted the pair-wise comparison over the predicted labels of exit flows and labels of Tor flows decided by the clustering algorithm. If the labels are the same in the correlated flows, they are TPs. If they are the same in the uncorrelated flows, they are FPs. Based on the preliminary

experiments, we decided to train k-NN models using Tor flows and further test them using exit flows. We also empirically chose k-means with ℓ_2 normalization to label the Tor flows.

Finally, we evaluated all methods using five *non-overlapping* windows, five seconds for the window interval, 5 for k, and 1 for ℓ. Based on Figure 6.2, the cosine similarity approach outperformed other methodologies. Since the DeepCoFFEA FENs were trained to maximize the cosine similarity between the correlated pairs and minimize the similarity between the uncorrelated pairs, the feature embeddings of the associated and non-associated flow pairs should reflect the margin in terms of the cosine similarity. We adopted this methodology when evaluating DeepCoFFEA throughout the remainder of the chapter.

6.2.3 FEN Architecture

As shown in previous work [5, 2, 1, 5], CNNs typically learn more useful features for analysis of Tor traffic. Thus, we further explored two different architectures, one based on 1D convolutional (Conv) layers and the other based on 2D Conv layers. We adopted the DF [2] and DeepCorr [5] architectures for these two architectures since they have been effective in generating website fingerprints based on traces between the client and the guard node and correlational flow features based on inflow and outflow to the Tor network. We empirically concluded that the 1D CNN-based DF architecture performed better; more specifically, we were unable to reduce the triplet loss below 0.01 with the DeepCorr architecture.

As shown in Figure 6.1, the two FEN models are learned, and in the testing phase, the A network maps inputs from Tor flows to feature embedding vectors, while the P/N network maps inputs from exit flows to feature embedding vectors. Each FEN consists of four 1D Conv blocks, including two 1D Conv layers, followed by one max pooling layer. After that, there was a fully connected output layer, which generated the feature embedding. We chose the input and output dimensions, optimizer, and learning rate based on parameter optimization as shown in Table 6.2.

6.2.4 DeepCoFFEA Evaluation Methodology

We outline an example DeepCoFFEA evaluation scenario in Figure 6.1 and detail each step as follows.

Window Partitioning.

Based on the DeepCorr dataset that we describe in detail in Section 6.3.1, we created the sets $T = \{t_1, ..., t_n\}$ and $X = \{x_1, ..., x_n\}$ of Tor and exit flows, respectively. Then, we computed statistics on the distribution of the number of packets and total flow duration of each flow to identify candidates for the number of windows k and the window interval length. Once these were chosen based on hyper-parameter tuning (Section 6.3.3), we divided each flow T and X into five windows, $W_1, \ldots, W_5$ of five seconds each in the scenario shown in Figure 6.1. To maintain the same dimension for all windows, $|W_1| = \cdots = |W_5|$, we padded the traces of shorter flows with zeros and truncated the traces of longer flows. Note that the sizes of the flow traces for windows in Tor and the exit traces are different since we used a partition strategy based on time. For example, if we used 5 seconds for the interval, the flow lengths for a Tor and exit flow collected in 5 seconds were different.

Even though we only showed non-overlapping window partitioning in this example, we further studied *overlapping* window partitioning to create windows which partially overlap each other. We will discuss it in detail in Section 6.3.2.

Training/Testing Set Partition.

We constructed the training and testing sets using windows. For example, when we used five windows, we had $[T_w]$ for Tor traces and $[X_w]$ for exit traces, where $w \in \{1, \ldots, 5\}$. We split them into one pair of training sets, $T_{tr} = \{T_1, \ldots, T_5\}$ and $X_{tr} = \{X_1, \ldots, X_5\}$; and five pairs of testing sets, $(T_{te1}, X_{te1}), \ldots, (T_{te5}, X_{te5}) = (T_1, X_1), \ldots, = (T_5, X_5)$. Note that the training data contained all windows while each of the five testing set pairs corresponded to an individual window.

In addition, if we had n training correlated flow pairs and m testing pairs, the FENs were trained only with n triplets, and each of the two trained FENs were applied to only m flows to determine the correlation of m^2 pairs of Tor and exit flows.

Training and Testing.

We trained FENs with T_{tr} and X_{tr} and tested models for each of the 5 testing sets (T_{tew},X_{tew}).

Evaluation.

For each testing window w, we generated feature embeddings for each $t_{i,w}$ and $x_{j,w}$ and computed the cosine similarity scores for all pairs of t and x. Then, we recorded the output to determine whether they were correlated (1) or not (0), which we call votes. To evaluate the aggregation of those votes over all of the k windows, we counted as TPs the cases where there were at least $k - \ell$ positive (i.e., 1) votes for correlated pairs and FPs the cases when there were at least $k - \ell$ positive votes for uncorrelated pairs.

6.3 Evaluation Details

In this section, we detail the experimental settings including the dataset, features, window partition, hyper-parameter optimization, and metrics to evaluate DeepCoFFEA in Section 6.4.

6.3.1 Input Preprocessing

Dataset.

We used the dataset collected by Nasr, Bahramali and Houmansadr [5] to evaluate the DeepCoFFEA attack. To the best of our knowledge, this is the most comprehensive flow correlation dataset collected on the Tor network and reflects the effect of different circuit usage and time gaps between training and testing data. Since we need to reconstruct the original packet traces based on the timestamps in which the outgoing and incoming packets were interleaved, we re-parsed the raw captures of DeepCorr set instead of using the preprocessed data in which the outgoing and incoming packets were separated. We further selectively chose the flow pairs to ensure that all connection destinations are unique, resulting in 12,503 flow pairs. This ensures that there should be no overlapping destinations between training and testing sets.

To show the impact of various experimental settings, we built diverse sets based on the DeepCorr dataset. First, to investigate the impact of diversity in the dataset

on the performance of DeepCoFFEA, we constructed two different sets, one (JAN) including only flows collected in January 2018; and the other (DIV) involving flows collected in January, February, and April, with more diverse circuits, to train and test DeepCoFFEA. We also included defended traces collected with the obfs4 in DIV to enable FENs to generate feature embeddings for both defended and undefended traces. Note that 10% of DIV comprises defended traces. For comparison, we also constructed an UDEF set consisting of only undefended flows. Second, to show the impact of different circuit usage in collecting the training and testing sets, we built a TWO set by filtering out flows collected using two arbitrary circuits in the DIV set. Then, we named the remaining flows in the DIV set as an ALL set. In this way, circuits to collect the TWO set are different from circuits to harvest the ALL set.

Window Pooling.

In contrast to DeepCorr, in which n correlated flow pairs could be used to create up to n^2 input pairs, in DeepCoFFEA each correlated flow pair can only produce at most one triplet, creating many fewer training examples for the FEN models. Instead of feeding all possible pairs to FENs, for each anchor point, we selected *one* semi-hard-negative that was more optimal through triplet mining. For example, if we have three flow pairs, (t_1,x_1), (t_2,x_2), and (t_3,x_3), our approach results in three input pairs such as $[(t_1,x_1,x_2)$, (t_2,x_2,x_1), $(t_3,x_3,x_2)]$. Based on our epoch generator, only first and third triplets can appear in the same epoch.

By partitioning the flows into k windows, we were able to pool the correlated pairs across windows, increasing the FEN training set size by a factor of k. More specifically, we divided each flow based on a predefined time interval chosen during the tuning (Section 6.3.3), and constructed the training set. Based on 10,410 pairs of training flows, we built a training set using all k flow windows, resulting in $k \cdot 10,410$ correlated flow pairs. In contrast, we constructed k testing sets separately for each window in which there are 2,093 pairs. Thus, this setup resulted in testing DeepCoFFEA using 2,093 Tor and exit flows across k windows. We detail the window partitioning in Section 6.3.2.

Features

As in DeepCorr [5], we used inter-packet delay (IPD) and packet size information to

Table 6.1: Mean total packet count per window.

Interval	2	3	4	5	6	7	8
Tor flow	68	105	137	171	224	266	273
Exit flow	42	65	85	106	139	166	169

construct the feature vectors from the flows. Since we chose 1D CNN models for DeepCoFFEA, we constructed one-dimensional arrays, $v_i = [I_i||S_i]$ for the bi-directional Tor and exit flows by concatenating the vector of IPDs and packet sizes. Here, the vector I_i consists of upstream IPDs (I^u) and downstream IPDs (-I^d) and the vector S_i is comprised of upstream packet sizes (S^u) and downstream packet sizes (-S^d).

We also evaluated DeepCoFFEA based on other combinations, such as $v_i = [I_i^u||S_i^u||I_i^d||S_i^d]$ or $[I_i^u||I_i^d||S_i^u||S_i^d]$ and these feature vectors led to much worse performance, perhaps because the local interleaving of upstream and downstream traffic allowed the (local) CNN filters to extract more relevant features. Similarly, we tried training FENs based on feature sets using only the IPD vectors or the packet size vectors and found that these were less effective as well.

Lastly, we evaluated DeepCoFFEA using feature vectors based only on packet sizes. The cosine similarity scores between correlated flow pairs were higher when considering both packet timing and size information.

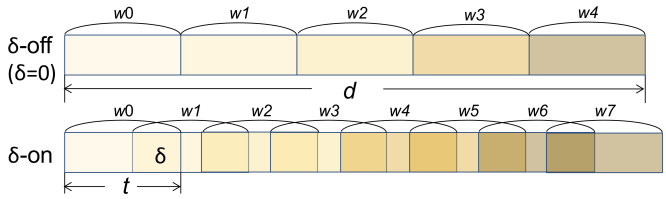


Figure 6.3: Overlapping window partition.

6.3.2 Window Partitioning

We tested two partition strategies, one based on time intervals and the other based on the number of packets, and then decided to use time intervals as the window *interval*, which yielded a better triplet loss curve. To determine the number of windows and the interval, we computed the total flow duration of Tor and exit traces in the DeepCorr

set. We found that the majority of the traces were captured in 20-40 seconds. To decide the most effective number of windows, we further computed the mean of packet counts for various intervals, 2-8 seconds (Table 6.1). To ensure that the packet count of the windows was greater than 100 for both Tor and exit flows, we had to choose intervals greater than 4 seconds. We finally chose 5-8 seconds as more promising intervals because they could yield enough windows (i.e., at least 5 windows) over 40 seconds.

We further explored overlapping window partitioning to create overlapping windows with some interval overlap (δ) between subsequent windows, which we refer to as δ-*on* partitioning. As shown in Figure 6.3, when the window interval length is t and total flow duration is d, then δ-*off* partitioning leads to $\lceil \frac{d-t}{t} \rceil + 1$ intervals, where window w is the interval $[t \times w,\ t \times w + t)$. In contrast, δ-on results in $\lceil \frac{d-t}{t-\delta} \rceil + 1$ intervals, where window w is the interval $[(t - \delta) \times w,\ (t - \delta) \times w + t)$. For example, when $t = 5$, $d = 25$, and $\delta = 3$, the 5 δ-off windows are the intervals [0,5), [5,10), ..., [20,25), while the 11 δ-on windows are the intervals [0,5), [2,7), ..., [20,25). As such, with δ-on, we create more windows, leading to more training flow pairs and boosting difference in TPs and FPs by aggregating results from more windows. In other words, δ-on increases the amplification of DeepCoFFEA, improving the performance dramatically, as demonstrated in Section 6.4.1.

6.3.3 Hyperparameter Optimization

We implemented FEN models using Keras [115] with Tensorflow [116] backend. The choice of hyperparameters is crucial to improve the DeepCoFFEA performance, and particularly the behavior of the FENs, to result in lower triplet loss. Thus, we explored parameter search spaces shown in Table 6.2 using training data of DIV set and one Nvidia RTX 2080 and one Tesla P100 GPUs. Note that we only considered the δ-off setting in Table 6.2 for more efficient optimization. Even though we also confirmed that all chosen parameters except window count behaved similarly in the δ-on setting using a small scaled DIV training set, the evaluation results in the δ-on setting may be improved with further parameter optimization.

First, the triplet loss aimed to separate the positive pair from the negative by a distance margin, α. We tuned α to maximize the distinction between the cosine similarity scores of the correlated pairs and those of the uncorrelated pairs. With α=0.1,

Table 6.2: Chosen hyper-parameters and search spaces used in the hyper-parameter optimization.

Param	Chosen Param	Search Space
α	0.1	$\{5 \cdot 10^{-2}, \ldots, 5 \cdot 10^{-1}\}$
Tor flow size	238	$\{97, \ldots, 300\}$
Exit flow size	140	$\{68, \ldots, 170\}$
Total flow duration	25	$\{20, \ldots, 45\}$
Window count	5	$\{4, 5\}$
Window interval	5	$\{5, 6, 7, 8\}$
Epoch generator	1	$\{1, \ldots, 10\}$
Output node	64	$\{10, \ldots, 100\}$
Optimizer	SGD	SGD, Adam
Learning rate	10^{-3}	$\{10^{-3}, \ldots, 10^{-4}\}$

the FENs attained the lowest loss.

Second, based on 25-40 second traces, we tested different combinations of the window count and window interval (e.g., 5 sec $\times$ 5 windows, 6 sec $\times$ 5 windows) to occupy at maximum 40 seconds, to be used as the search space. We then empirically found that t=5 and k=5 yielded the best results and computed the minimum and maximum number of packets corresponding to 5 seconds based on the Tor and exit flows. Finally, this analysis resulted in 97 and 300 packets for Tor flows and 68 and 170 packets for the exit traces. We used these numbers for the FEN hyperparameter search spaces, as shown in Table 6.2.

Third, as discussed in Section 6.2.1, we implemented our own triplet epoch generator which selected triplets for the positive and negative networks from separate pools. We further tuned the number for how frequently those separate pools needed to be updated (i.e., shuffled and divided into two pools). Eventually, the FEN performance improved when updating the pools more frequently. Thus, we recommend updating them every epoch rather than every 2-10 epochs.

Finally, we further tuned the learning rate of SGD optimization and the number of output nodes, which is the dimension of the feature embeddings generated by the trained FENs.

6.3.4 Metrics

In this section, we introduce the definitions of the TPR, FPR and BDR metrics used in Section 6.4 and highlight how TPR and FPR in our metrics are different from those used by DeepCorr [5].

First, we review the standard TPR, FPR, and BDR and then, discuss how we computed them to incorporate the results for multiple windows.

- TPR: The true positive rate is the fraction of correlated flow pairs that are classified as "correlated".
- FPR: The false positive rate is the fraction of uncorrelated flow pairs that are classified as "correlated".
- BDR: The Bayesian detection rate is not measured by state-of-the-art attacks; however, this metric represents the feasibility of attacks since it considers the base rate (or prior). For example, even with a high TPR and low FPR, the success rate of the correlation attack would be very low if the prior was low. BDR in the flow correlation is the probability that a correlated pair is actually "correlated" given that the correlation function detected it as "correlated" and it can be computed as follows. Compared to the precision rate, BDR is more directly impacted by the base rate.

$$P(P|C) = \frac{P(C|P)P(P)}{P(C|P)P(P) + P(C|N)P(N)}$$

  where $P(P)$ is the probability that the pair is correlated, $P(C)$ is the probability that the pair will be decided by the flow correlation as correlated, and $P(C|P)$ and $P(C|N)$ are replaced with TPR and FPR, respectively.

As outlined in Section 6.2.4, we tested DeepCoFFEA by aggregating predictions for each window. To compute the TPR and FPR, we vote for each window of all flow pairs based on the cosine similarity scores and the threshold, τ. That is, we vote with 1 if the similarity score is greater than τ and 0 otherwise. If the flow pair obtained at least $k-\ell$ 1 votes, we counted the pair as a TP for correlated samples and as FP for uncorrelated samples.

In addition, we measured the performance of state-of-the-art attacks and DeepCoFFEA using ROC curves by varying the correlation threshold parameter. Note that we also plotted these curves for RAPTOR and DeepCorr and computed the TPR, FPR, and BDR for our evaluation set while varying the tradeoff parameter for these attacks. Note that while the parameter has different units for all three attacks – a Spearman correlation coefficient for RAPTOR, sigmoid output for DeepCorr, and cosine similarity in DeepCoFFEA– we still see a smooth TPR/FPR tradeoff characteristic. We further discuss how we chose the thresholds for DeepCoFFEA in Section 6.3.5.

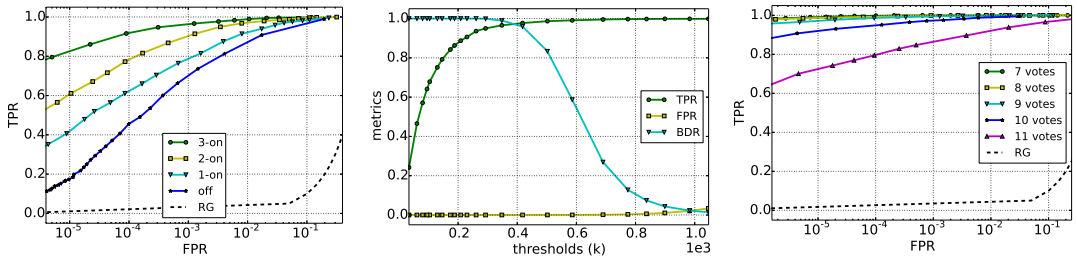
6.3.5 Thresholds

In DeepCoFFEA, the embedded feature correlation threshold τ acts to control the number of exit traces that are classified as "possibly correlated" with each Tor trace in a given time window. We can either control this number *indirectly* by setting a *global* threshold τ that is applied to all cosine similarities (so that, generally, as τ increases, fewer pairs will be classified as possibly correlated); or we can control this number *directly* by classifying only the closest κ exit traces to a given Tor trace t_i (as measured by $cos(G(t_i), H(x_j))$) as possibly correlated. This latter choice corresponds to computing a *local threshold* for each t_i by sorting the list $d_{i,1}, ...d_{i,n}$, where $d_{i,j} = cos(G(t_i), H(x_j))$, and selecting the κ^{th} element as the threshold for t_i.

We explored both approaches and empirically found that, by directly controlling the rate of positive results, the local threshold approach yields a better ROC curve. As shown in Figure 6.4b in Section 6.4, the TPR and FPR per window are directly proportional to κ. The ROC curves of DeepCoFFEA in Section 6.4 were generated using κ as the curve parameter.

6.4 Evaluation Results

In this section, we evaluate the overall performance of DeepCoFFEA using the experimental setup and metrics discussed in Section 6.3.4. We explore various configurations of DeepCoFFEA to find the most effective setting in Section 6.4.1. Then we compare the effectiveness and efficiency of DeepCoFFEA to state-of-the-art attacks in Section 6.4.2.



(a) ROC for δ-on/off (loss $\approx$ 0.02).

(b) TPR,FPR, and BDR with varying positive counts κ whendloss $\approx$ 0.006.

(c) ROC with $k = 11$ windows and $\ell = 0, 1, 2, 3, 4$ (loss $\approx$ 0.006).

(d) ROC at various triplet training losses.

(e) ROC for FENs trained using DIV, JAN, APR, and UDEF datasets (loss $\approx$ 0.01).

(f) ROC to show the impact of defended traces and testing set circuits (loss $\approx$ 0.006 for obfs4).

Figure 6.4: Performance of DeepCoFFEA across various settings (Note that RG is Random Guess and all x-axes except Figure 6.4b are log scale).

6.4.1 DeepCoFFEA Performance

Window Partition

First, we investigated the impact of δ-on/off settings for varying δ. As discussed in Section 6.3.2, δ-on creates $\lceil \frac{25-5}{5-\delta} \rceil + 1$ windows, that is, it creates 6 windows for $\delta = 1$, 8 windows for $\delta = 2$, and 11 windows for $\delta = 3$. Note that we omitted $\delta = 4$ since with 21 windows the resulting cosine similarity matrix for 218,610 training flow pairs was too large to compute using our resources, so we could not select semi-hard-negatives.

We reported results of $\ell = 2$ for the 3-on and the 2-on and $\ell = 1$ for the 1-on setting in Figure 6.4a. The overlapping windows clearly made DeepCoFFEA more effective. Furthermore, since increasing δ further improved the amplification capability with more training flow pairs as well as more voting results, Figure 6.4a shows that a larger δ led to superior ROC curves with higher TPRs using more votes. This also indicates that more resourceful adversaries could further improve the 3-on results by using the 4-on setting. Based on Figure 6.4a, we evaluated DeepCoFFEA in the 3-on setting throughout the remainder of the section.

Threshold Parameter

Using the experiment setup described in the previous section, we evaluated the effect of the positive correlation parameter κ as described in Section 6.3.5. Figure 6.4b shows the results of this evaluation and the overall performance of DeepCoFFEA. Both the TPR and FPR increased by increasing the threshold, while the BDR decreased. We note some points of interest on the curve: for 10^{-3} FPR, the TPR was 0.995; for $3 \cdot 10^{-4}$ FPR, it was 0.991, and for 10^{-5} FPR the TPR was 0.977. DeepCoFFEA successfully reached 0.95 TPR with only 2 FPs and 0.94 TPR without FPs (i.e., 0 FPR). These results indicate that DeepCoFFEA achieved almost perfect performance correlating the Tor and exit flows.

Vote Threshold.

After computing the cosine similarity scores for all testing pairs for each window, we decided that the flow pair would be correlated if it had at least nine 1 votes across 11 windows. Figure 6.4c, shows how the performance changes when requiring positive correlation in 7-11 windows.

Triplet Loss.

When training FENs, the triplet loss started with 0.09-0.1 and decreased monotonically with training time; our experiments halted training when the loss hit 0.006. In this section, we investigate the performance of DeepCoFFEA when training stops at different loss values; this experiment gives insight on choosing a stopping point for training FENs. Based on Figure 6.4d, the DeepCoFFEA performance continually improved as the triplet loss decreased even though the degree became insignificant after the loss reached 0.008.

Dataset.

Previous related work [112, 10] required very large data sets for training and testing. To study the impact of the dataset diversity on FEN training, we used the DIV, and JAN sets to train and test FENs. As Figure 6.4e shows, the performance of DeepCoFFEA was improved with the DIV set in which more variance between Tor traces existed because the traces were collected with more circuits for a longer period (i.e., four months) and some traffic with the obfs4 defense. However, even the DIV set is much smaller than the training set used in TF. For example, the FENs were trained using 114,510 triplets every epoch while TF [10] was trained with 232,500 triplets. This indicates that DeepCoFFEA performance could be further improved with a much larger dataset.

Since we included defended flows in the DIV set, we further investigated the UDEF set to study how this affected the performance of DeepCoFFEA. Based on Figure 6.4e, when we trained and tested DeepCoFFEA using only undefended traces, this change minimally impacted the performance. In contrast, as shown in Figure 6.5, the DeepCorr degraded using the DIV set because the intra-class variances became larger with defended flows. Since DeepCoFFEA does not learn based on labels, the inclusion of defended traces added more diversity to the set, which enabled DeepCoFFEA to broaden the learning scope to better distinguish correlated flows from uncorrelated flows.

Testing Set.

TF [10] could be applied to data collected from a different distribution from the training set; in particular they used a TF model trained on one data set to successfully identify website traces collected on different data sets collected up to three years apart. We explored the same goal for correlation attacks using different testing sets. Using a training set collected in January 2018, we evaluated the performance of DeepCoFFEA

on a testing set collected in the same month, the JAN set, and on a testing set collected three months later, the APR set. As shown in Figure 6.4f, both experiments presented comparable performance.

Circuit Impact.

We investigated the effect of circuits used in the training and testing sets. This experiment helps understand how much DeepCoFFEA successfully detects correlated flows which were collected using *arbitrary* circuits. We trained FENs using training data of the All set. As shown in Figure 6.4f, when we used the TWO set as testing flows and the loss value was 0.02, the performance became slightly less effective although the discrepancy was not critical. However, as the triplet loss decreased, both TWO and ALL sets showed comparable performance, indicating that DeepCoFFEA still successfully detected correlated testing flows collected using arbitrary circuits.

Defended Traces.

Nasr, Bahramali and Houmansadr [5] evaluated DeepCorr against traces protected by the obfs4 pluggable transport (PT), the PT recommended by the Tor project [71]. obfs4 encrypts and transforms the traffic between the client and the guard node to avoid potential traffic analysis-based censorship. In particular, it obfuscates packet sizes by appending random padding. To add extra security, obfs4 also provides an IAT (Inter-Arrival Timing) mode that randomizes inter-arrival times, which is enabled when IAT mode = 1 (obfs4-iat1) and disabled when IAT mode = 0 (obfs4-iat0). In this study, we explored the obfs4 PT with both modes as shown in Figure 6.4f.

Note that we trained FENs using the DIV set and tested them using the obfs4-iat1 and obfs4-iat0 flows. Even though the performance was worse than undefended flow correlation, DeepCoFFEA still performed better than random guessing with 0.76 TPR when FPR was 10^{-3} against obfs4-iat0. In particular, even against obfs4-iat1, DeepCoFFEA achieved 0.61 TPR with 10^{-3} FPR, which indicates that the obfs4 PT did not defeat DeepCoFFEA completely and Tor must be equipped with more robust defenses against DeepCoFFEA-style correlation attack.

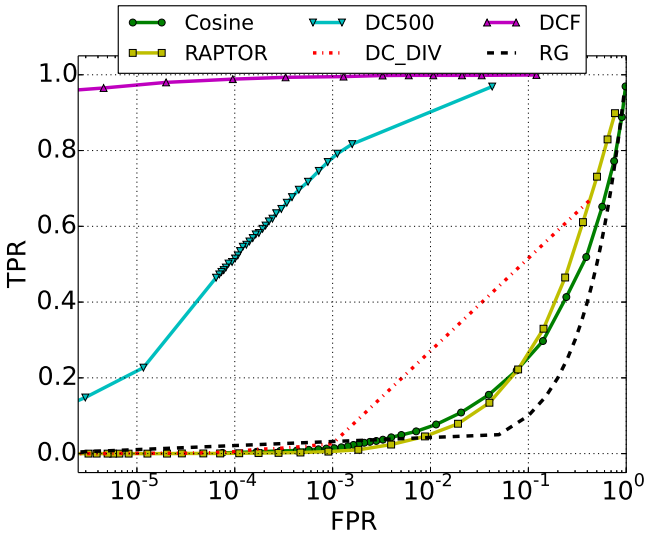


Figure 6.5: ROC of state-of-the-art and DeepCoFFEA attacks (Note that x-axis is log scale, **DC**: DeepCorr, **DC-DIV**: DeepCorr with the DIV set, **DCF**: DeepCoFFEA (loss ≈ 0.006), and **RG**: Random Guess.)

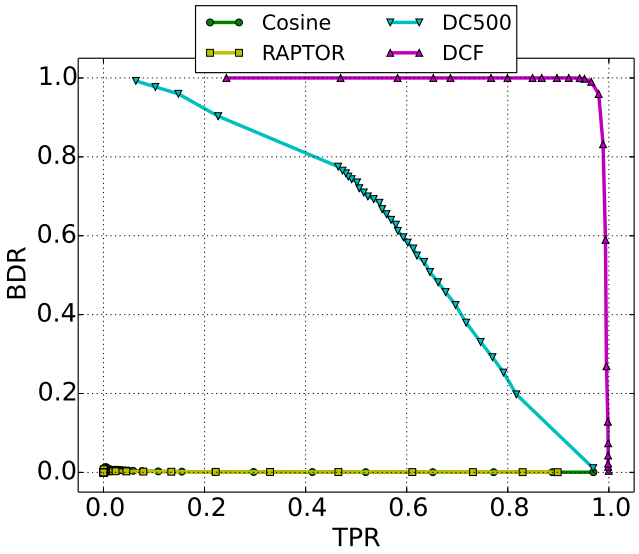


Figure 6.6: BDRs of state-of-the-art and DeepCoFFEA (loss ≈ 0.006) against TPRs (Note that x-axis is log scale, **DC**: DeepCorr, and **DCF**: DeepCoFFEA (loss ≈ 0.006)).

Table 6.3: The number of packets for each of 11 windows (Total flow duration: 25 seconds).

window	0	1	2	3	4	5	6	7	8	9	10	total
Tor	100	126	169	211	244	261	261	251	237	225	231	1,148
Exit	77	101	130	150	162	162	157	148	140	137	140	672

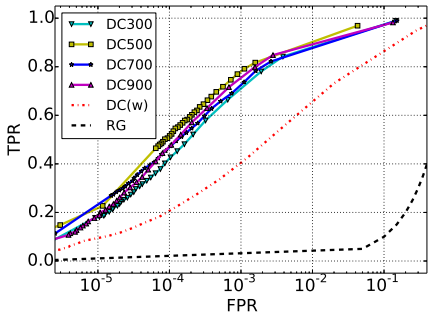


Figure 6.7: ROC of DeepCorr (DC) by varying the flow length (i.e., the number of packets) (Note that x-axis is log scale and **DC(w)** is when evaluating DeepCorr in the window setting).

6.4.2 Comparison to State-of-the-art

In this section, we compare the performance of DeepCoFFEA to other state-of-the-art attacks using 2,093 flow pairs of the DIV set.

Tuning State-of-the-art.

For a fair comparison, we need to tune each attack to obtain the best performance on our evaluation set. First, we empirically chose the best feature dimension (number of packets) for all three attacks and additionally determined the number of training flow pairs for DeepCorr. The dimension is important because all three prior attacks require both Tor and Exit flow feature vectors to have the same length, and using longer vectors will induce padding that decreases accuracy, while using shorter vectors might truncate useful information. Through experimentation, using 900 packets for Cosine Similarity and RAPTOR gave the best performance; the corresponding parameters used for DeepCoFFEA are shown in Table 6.3.

However, for DeepCorr, we found that the performance became notably worse using the DIV set due to the defended traces and better when trained and tested with the preprocessed feature vectors included with the released DeepCorr data set. Thus, we

Table 6.4: TPRs of DeepCorr (**DC**) and DeepCoFFEA (**DCF** when loss ≈ 0.006) by fixing FPRs(#FPs) when tested with 2,093 Tor connections resulting in 2,093 correlated and 2,093 $\times$ 2,092 uncorrelated flow pairs.

FPR	$10^{-3}(4400)$	$3 \cdot 10^{-4}(1300)$	$9 \cdot 10^{-5}(400)$	$2 \cdot 10^{-7}(1)$
DC	0.791	0.662	0.506	0.06
DCF	0.995	0.993	0.989	**0.942**

Table 6.5: Time complexity (seconds) of DeepCorr (**DC**) and DeepCoFFEA (**DCF**) by varying the size of testing flow pairs.

Size	100	1,000	5,000	10,000
DeepCorr	56	5,438	137,000	549,000
DeepCoFFEA	2	9	150	584
DC:DCF	28:1	604:1	913:1	940:1

decided to use their preprocessed set. We further noticed that flow pairs in which one flow required a large amount of padding reduced the accuracy of DeepCorr, so to make the most optimistic estimation of the DeepCorr performance, we further selectively chose the 5,000 training and 2,093 testing flow pairs that minimize the total padding amount. We also stopped training if the testing TPR and FPR did not improve for a day.

Based on Figure 6.7, we decided to use 500 packets as the flow length since the performance became more effective than other flow lengths. If we use a smaller number of flow pairs in which we can further reduce the amount of padding for all flows, the performance may improve with an increased feature dimension; however, this would not reflect the variable length of Tor flows in the real world.

DeepCorr with DIV.

As shown in Figure 6.5, DeepCorr became much less effective using the DIV set because the inclusion of defended traces led to much higher intra-class variances for both labels (i.e., correlated or uncorrelated). Such diversity in the training data weakened the learning capability of label-based, supervised DeepCorr models while it improved the feature extraction ability of FENs.

Overall Performance.

The results of our comparison are shown in Figures 6.5 and 6.6. DeepCoFFEA outperformed all other correlation attacks, reaching a much higher TPR for any given FPR. This substantial improvement correspondingly led to a higher BDR. As shown in Figure 6.6, DeepCorr and DeepCoFFEA were both able to achieve a BDR of 1.0 for our small evaluation set, while RAPTOR and cosine similarity were unable to reach a FPR of 0 with a positive TPR. However, for BDRs closer to 1, the TPR of DeepCorr degrades considerably while DeepCoFFEA can still detect most associated pairs correctly (i.e., more than 94% of the entire set). For example, to exceed 99% BDR, DeepCorr must use a threshold so high that TPR dropped to 6%, while DeepCoFFEA was still able to achieve 95% TPR. Further comparisons between the tradeoffs among TPR, FPR, and BDR for DeepCorr and DeepCoFFEA are shown in Table 6.4. DeepCoFFEA outperformed DeepCorr by significant margins, +20-88%, at fixed FPRs.

Previous literature [5] suggested that DeepCorr might be applied in a multi-stage attack in which the correlation analysis is first done in N flow pairs using the first p packets and then, extended to only M flows (i.e., $M < N$) identified as correlated in the first phase of attack using $p + l$ packets. In this way, DeepCorr might improve BDRs due to the increased base rate. However, as shown in Figure 6.7, in our experiments, longer flow observations (i.e., 700 and 900 packets) did not yield better performance in DeepCorr, which indicates that this hierarchical concept will not achieve better TPRs and FPRs than one-stage attack using 500 packets. Note that using a 7,093 flow pair set, the performance did not improve after 500 packets. Hence, it remains unclear whether a multi-stage approach can fundamentally resolve the low BDR issue.

To sum up, the combination of feature embedding and amplification help to improve the state-of-the-art performance significantly, to the point that a DeepCoFFEA-based end-to-end correlation attack may be feasibly deployable at Tor scale.

Time Complexity.

We further compared the running time of DeepCorr and DeepCoFFEA by varying the testing set size (t_n) from 100 to 10,000 flow pairs. We used one Tesla P100 GPU with 16GB memory and 900 packets for DeepCorr. We computed the total running time to complete the full $t_n \times t_n$ flow attack including data loading and the correlation metric

computation; for DeepCoFFEA, we measured the total time for loading testing flows, generating feature embeddings, computing cosine similarity scores, and aggregating the resulting votes across 11 windows.

We report the mean value of five runs for each attack in Table 6.5. Compared to DeepCorr, DeepCoFFEA achieved much lower computational costs, in particular, to conduct the correlation analysis for 10,000 flow pairs, DeepCoFFEA performed around 940 times faster than DeepCorr. We also note that the cost gap between DeepCorr and DeepCoFFEA further increased as the flow pair count increased.

There are two reasons for this discrepancy. First, we only needed to evaluate the Tor FEN $11 \times t_n$ times and the Exit FEN $11 \times t_n$ times to generate all feature embeddings, rather than evaluating t_n^2 instances of the DeepCorr CNN. Second, even though the similarity score computation over the generated t_n^2 embeddings became more expensive for larger testing set, we further optimized the cosine similarity computation by converting the embedding vectors into matrices, and using the vectorized routine of `sklearn.metrics.pairwise.cosine_similarity`.

Even though the multi-stage setting which we discussed earlier can somewhat reduce the complexity overhead of DeepCorr for longer flows, we found that the complexity of correlating shorter flows was still considerable. For example, when we applied a DeepCorr model optimized for the first 300 packets to correlate 10,000 flow pairs, the attack still took $2.2{\cdot}10^5$ seconds for this first stage, or more than two orders of magnitude more computation than DeepCoFFEA. Moreover, the performance improvement by a multi-stage attack is somewhat doubtful. As mentioned previously and shown in Figure 6.7, we found that the increased padding with longer flows offsets the potential accuracy gains for the smaller number of flows that reached this length. Hence, even with this hierarchical approach, it is hard to expect that DeepCorr can achieve either better performance or lower computational cost than DeepCoFFEA.

Time Gap between Training and Testing Set.
As discussed in Section 6.4.1 and shown in Figure 6.4e, DeepCoFFEA perfectly detected whether or not a flow pair was correlated when using a testing set (APR) with a different distribution from the JAN training set (0.99 TPR and 10^{-3} FPR). In contrast, the TPR of DeepCorr decreased by 50% at a FP of 10^{-3} [5].

Table 6.6: DeepCorr and DeepCoFFEA performance against obfs4 pluggable transport.

Attack	IAT = 0		IAT = 1	
	TPR	FPR	TPR	FPR
DeepCorr	0.50	0.0005	0.10	0.001
DeepCoFFEA	0.69	0.0005	**0.61**	0.001

Table 6.7: DeepCorr performance for each window when κ=34 (Note that pkt# is the packet count and TPRs/FPRs (%)).

Window	0	1	2	3	4
Pkt#	100	200	238	238	238
5 seconds	69/2	53/2	32/2	10/2	7/2
Pkt#	100	100	100	100	100
100 packets	87/5	16/2	5/2	5/2	5/2

This indicates that the FENs used in DeepCoFFEA require less frequent re-training than the DeepCorr CNN. We speculate that this is because the triplet loss function gives the FENs a significantly higher degree of freedom in separating correlated and uncorrelated pairs than the supervised learning loss function required in DeepCorr. This unsupervised learning process seems to lead to less overfitting to the training set and result in more generalized FEN models. We leave further investigation using much older testing sets as future work.

Robustness against Defense.

Figure 6.4f shows that DeepCoFFEA significantly outperformed random guessing even against obfs4-iat1. We further compared this result to DeepCorr. As shown in Table 6.6, DeepCoFFEA outperformed DeepCorr by 19% in obfs4-iat0, and by 51% in obfs4-iat1. This indicates that DeepCoFFEA has superior performance to DeepCorr against obfs4-protected traffic and further, does not require tuning and training new models better tailored to detect correlated defended flows; thus, we can use FENs for both undefended and defended correlation studies.

6.4.3 Amplification in DeepCorr.

Amplification, or window partitioning, is a key technique to reduce the number of FPs in DeepCoFFEA. In this section, we investigate whether this strategy could also be

applied to DeepCorr, e.g. whether amplification can be applied independently of feature embedding to improve the performance of end-to-end correlation attacks. To provide a fair comparison, we applied the *local* threshold approach as explained in Section 6.3.5. Even though we only explored the δ-off setting in this experiment, we do not expect the performance improvement with the δ-on settings because DeepCorr performed poorly on all windows except the first window, which will be discussed in detail later.

First, we evaluated DeepCorr in the same window setting (i.e., amplification) using five windows, five seconds per window, and 238 for the flow length. The only difference with DeepCoFFEA setting is that the DeepCorr architecture requires using the same number of packets for both Tor and exit flows, which requires more padding or truncation of feature vectors (Refer to Table 6.3). We found that this loss of detail highly reduced the accuracy of the DeepCorr metric, resulting in an accuracy for each window of less than 10%.

Since the length of traces was more consistent within individual windows, we also tried to train separate DeepCorr models per window, finding the optimal flow length for each window. However, this Amplified DeepCorr still performed worse than the original DeepCorr settings shown in Figure 6.7. When investigating each window result separately, Table 6.7 shows that the performance became worse in later windows, in particular, TPR degraded quickly enough that very few flow pairs would get a 1 vote in windows 2, 3, or 4. This is also consistent with our observation that the DeepCoFFEA embedded feature vectors for earlier windows were generally more highly correlated than those in later windows, although the drop was not as extreme as seen here.

Finally, we also tried partitioning flows into windows based on 100-packet segments, however, this resulted in much worse performance than the previous scenario. In particular, DeepCorr attained lower TPR ($\approx$ 5-16%) for windows 1-4 even though the first 100 packets had shown stronger correlation based on Table 6.7. This is because after the first window, the start packets of the two flows have different timestamps, thus the order of packets in subsequent windows carry less and less useful information about the correlation between flows.

Note that this performance could be improved by tuning each window model more thoroughly, however, it is still obscure how much DeepCorr could benefit from amplification due to two factors. First, the number of packets in a specific duration varies highly

by flow, meaning any window-based approach would result in frequent padding and truncation of flows, substantially impacting the performance of the DeepCorr model. Second, the subflows in a correlated pair become more varied in later windows, resulting in more considerable intra-class variance and a lower TPR. These limitations are significant obstacles to training supervised models.

We summarize the benefits of DeepCoFFEA compared to DeepCorr as follows.

- The pairwise nature of correlation metrics used in DeepCorr resulted in low BDRs and required high computational complexity to achieve minimally effective performance. DeepCoFFEA illustrates that these limitations do not preclude real-world correlation attacks on Tor with high BDRs and much lower time complexity.

- Our new attack framework — incorporating multiple DNNs (FENs), trained using triplet loss and window partitioning — enabled us to achieve a lower computational cost by extracting only $O(n)$ feature embedding pairs of Tor and exit flows based on FEN models. Furthermore, this new framework significantly reduced the number of FPs, by aggregating votes from multiple windows.

- DeepCoFFEA performed effectively against a testing set collected three months later than the training set. This result demonstrates that FEN models can be reused for a short-term period, such as three months, which requires *less frequent re-training* than DeepCorr, which requires updating models every three weeks.

- The obfs4 traces failed to defeat DeepCoFFEA completely. Even in the highest security setting, DeepCoFFEA still significantly reduced the anonymity set, with 0.61 TPR while DeepCorr achieved 0.10 TPR to yield 10^{-3} FPR. Furthermore, DeepCoFFEA can conduct flow correlation analysis for both undefended and defended traces without training separate models.

6.4.4 Countermeasures

In this section, we discuss possible countermeasures to thwart DeepCoFFEA-style attacks. Figure 6.4f demonstrates that the obfs4 defense mechanism with IAT mode = 1 degraded the DeepCoFFEA performance by around 39% margin at 10^{-3} FPR compared to undefended flow correlation. As such, we recommend using obfs4-iat1 as an initial

defense. Relay selection algorithms [38, 31, 40, 41] designed to hinder potentially malicious ASes from monitoring both ends of Tor connections may also help to deteriorate the effectiveness of DeepCoFFEA as prior work [5] also pointed out.

We expect that, if implemented, higher-overhead traffic analysis countermeasures such as BuFLO [15] and Tamaraw [16] would be effective against DeepCoFFEA attacks as well. Since both defenses hide the total packet statistics by padding dummy packets until the flows reach their thresholds such as the total number of packets or transmitted bytes, they can make Tor flows much less distinguishable from each other, which may further make correlated flow features less effective. However, given the high bandwidth overhead of the constant rate padding, it could be more interesting to explore the effectiveness of lighter-weight website fingerprinting defenses such as WTF-PAD [17] and Walkie-Talkie [79] in defeating flow correlation. Since these defenses have never been evaluated against triplet network-based attacks, adjusting them to defeat DeepCoFFEA is a compelling topic for future work.

Chapter 7

Conclusion and Future Research

In this thesis, we introduced various types of traffic analysis techniques that can be applied to achieve several more sophisticated fingerprinting goals by discovering more fine-grained features or adopting more advanced deep learning techniques.

First, we described a novel attack, keyword fingerprinting, to identify search engine queries over Tor, using new feature sets focusing on incoming packets in the response portion of a search query trace. We performed feature analysis to select appropriate new features for this classification task and analyzed the effect of several variations on the attack, including the choice of classifier, size and contents of the monitored set, the size and contents of the background training set, and the search engine and query method. Across these variations, the results show acceptable performance and suggest that new work is needed to understand how to defend against keyword fingerprinting attacks. This work is critical given the importance of protecting the contents of search engine queries.

Second, we extensively explored the effectiveness of DNNs in three different applications: automated feature engineering, fingerprinting attacks, and prediction of fingerprintability. As a feature extractor, lower dimensional representations learned by an AE have made state-of-the-art WF attacks more effective as well as more efficient. For fingerprinting attacks, DNNs have performed well across various traffic datasets and different fingerprinting tasks, as well as against recent WF defenses. Lastly, we showed that several features of the HTML-level design of a website influence the fingerprintability using DNN models. This finding shows the possibility that future work on WF

defense can use HTML features.

Third, we introduced a novel attack, GANDaLF, using GANs in a semi-supervised setting, in which the generator minimizes the difference between real trace and fake trace distribution. The discriminator is trained to distinguish between real and fake samples and further improve classification over the labeled set by leveraging both labeled and unlabeled traces. Because it requires only a small amount of labeled data, we investigated the applicability of this variant of GANs in a low-data setting for WF attacks. Furthermore, we evaluated GANDaLF by exploring the index and non-index pages of both sites using various experimental scenarios. Finally, our empirical study showed that GANDaLF had better performance than Var-CNN and TF, the most recent low-data WF attacks at non-index fingerprinting with a particularly significant performance advantage in the open-world setting. However, in WF-S, GANDaLF was not more effective than k-FP by leveraging the total packet statistics.

Fourth, we developed a new end-to-end flow correlation attack on Tor, DeepCoFFEA, which is more scalable and practically effective than state-of-the-art attacks. First, we explored the primary challenge in applying prior work to large-scale traffic analysis, low BDRs, due to the pairwise nature of flow correlation attacks. We then developed DeepCoFFEA to reduce FPs, leading to higher BDRs. By extending the triplet network so it is a suitable feature extractor for amplified flow correlation, we developed a pair of FENs that were jointly trained using the triplet loss function. By evaluating DeepCoFFEA in various experimental settings, we demonstrated that this new architecture and attack paradigm significantly improves state-of-the-art flow correlation attacks at the cost of acceptable time complexity.

In the next section, we introduce several future research directions for traffic analysis defenses and GANDaLF and DeepCoFFEA projects.

7.1 Future Work

7.1.1 Traffic Analysis Defenses.

Defense Evaluation.

In the last decade, security researchers have developed several types of WF defenses including constant padding, adaptive padding, and supersequence-based padding. In particular, defenses based on periodical padding [15, 16] are straightforward and powerful mechanisms in terms of increasing the number of false positives; however, they struggle with high enough bandwidth overhead to impact the service. In an effort to design more efficient padding, several researchers have adopted adaptive padding to develop relatively lighter-weight defenses such as WTF-PAD [17] by sending dummy packets only when it detects noticeable large gaps between packets. However, DL-based WF studies [2] have successfully defeated this defense with 90% accuracy. Later, other types of padding that mimic *supersequence* (i.e., representative traffic in each group of websites) have made website traces similar enough to each other to only allow up to a 50% detection rate. However, this defense leaked the timing information, and thus was successfully detected by timing-based WF attacks [97].

These weaknesses have become a major hurdle for the Tor project to deploy defenses in Tor. Recently, a more sophisticated approach to hide burst patterns was introduced by FRONT [117] and RegulaTor defenses [118]. To defeat DF, the former adds more randomness in the location and volume of dummy packets while the latter regulates burst shapes and sizes by sending dummy packets to incoming bursts and then decaying the packet sending rates. However, the extent to which these misclassification rates and bandwidth and latency overheads are consistent when they are implemented in the real-world is questionable. Therefore, similar to Juarez et al., who conducted this type of analysis against WF attacks [52], researchers need to evaluate defenses against realistic WF scenarios such as webpages and multi-tab fingerprinting and various client location and circuit usage. This study will help the Tor project and users better understand the practicality of recent WF defenses.

New Defenses.

As shown in both FRONT and RegulaTor, sending or delaying dummy packets located

in selective burst locations is a good strategy to weaken WF performance, even though it comes with some overhead costs. However, given that both defended traces still somewhat expose packet timing information [118], further study to hide the timing pattern seems to be an interesting future work.

For the other types of defense, efforts to extend our study in Section 4.5 are needed including more extensive fingerprintability analysis based on HTML features using thousands of Alexa top websites to locate more impactful website design properties. This could enable us to develop tools implemented in either of the client-side or server-side to further inspect websites and suggest safer designs for traffic analysis.

In addition, our study in Chapter 6 requests an urgent need to develop defenses to defeat DeepCoFFEA-style attacks are also needed. Since we found that DeepCoFFEA became much less effective with the packet count-based window partitioning, further investigation on the way to efficiently pad or delay packets in each window to have similar packet counts could be an important next step to design the defense.

7.1.2 More Rigorous GANDaLF Experimental Settings

Better WF-S Setting for TF and GANDaLF.
Since we used website fingerprints rather than subpage fingerprints as unlabeled data fed into the discriminator in the WF-S scenario, our experimental results in Table 5.5 were not based on the best setting. A larger-scale subpage set in terms of the number of websites, subpages, and training instances per subpage will lead to better accuracy. In particular, it will generate better fake subpage samples based on the feature of matching the loss to subpage data, which will help the discriminator classify more variable subpages as the correct class.

Similarly, this approach may improve the quality of the feature extractor of TF [10] by being pre-trained using a much larger subpage set, because the triplet network can better learn to distinguish between different websites based on subpage traces.

Incorporate Categorical Features.
As shown in Table 5.5, k-FP [88] had very good WF-S classification accuracy based on all of the statistics-based features. This result suggests a promising future research approach investigating how to aggregate these features into GANDaLF . In our preliminary

experiments, we confirmed that directly using k-FP features did not help improve the performance of GANDaLF. Thus, it may be achieved by an ensemble classifier based on k-FP (or other optimal classifier) with these categorical features, and GANDaLF, which aggregates the prediction decisions from the models. As shown with Var-CNN [11], this ensemble classifier may lead to enhanced classification performance.

Different Generator.

We can improve the ability of the generator to produce better fake samples. We may incorporate conditional GANs [119] into the SSL. Since the generator can be conditioned on some extra label information, we can generate fake trace samples conditioned on specific class labels. Thus, based on the discriminator classification result, we may force the generator to generate samples for target websites that produce more confusion during classification. Additionally, we can improve the generator by exploring other generative networks such as an autoencoder and variational autoencoder [120]. Moreover, building more traffic-specific generators may be an interesting future direction. This approach may be more tuned to create better fake trace samples to conduct better WF. For example, we may perturb burst patterns that are similar to cropping, resizing, and rotating images to create better fake samples that will be validated by a target classifier such as DF. That is, we can incorporate this generator into the SSL framework and maximize the classification ability by training the generator and classifier jointly.

Other WF Scenarios.

As we briefly mentioned in Section 5.1.2, the subpage identification challenge is an interesting scenario that has yet to be explored in detail. This setting poses a unique challenge in that classifiers must learn to handle data that have a wide range of inter-class variation. More precisely, classifiers must be able to distinguish between pages from different domains (high variance) as well as pages within the same domain (relatively lower variance). It would be interesting to investigate this new task in a low-data setting. Moreover, we can apply GANDaLF to resolve other unrealistic assumptions, as identified by Juarez et al. [52]. For example, GANDaLF may be aptly suited to detect websites that are visited from different versions of the Tor Browser Bundle. In this scenario, GANDaLF, when trained on a large variety of unlabeled data, can learn more robust decision boundaries. since the generator can generate fake samples to fill the

gap between the bundle versions. However, further architectural optimization is needed to elaborate on such feature generation when the intra-class variance is expected to be more significant.

7.1.3 More Realistic DeepCoFFEA Evaluation

Evaluation against WF Defenses.
The most pressing open question is whether the adaptive padding [25] defense under development by the Tor project will be an effective countermeasure to DeepCoFFEA style attacks. Given recent results showing that DNNs can defeat similar countermeasures such as WTF-PAD [2] and Walkie-Talkie [97] in the context of website fingerprinting, we should further investigate what mechanisms *can* be effectively deployed against DNN-based traffic analysis attacks. Our results suggest that these may be the most important questions anonymity researchers and developers currently face.

Even though WF defenses can be utilized to defend the end-to-end flow correlation attacks, defenses that are more specific to the correlation attacks have not been actively explored. Further study to adjust the padding mechanism to hide the traffic pattern in time-based windows is also an interesting future research direction to propose more powerful defenses against DeepCoFFEA-style attacks.

More Advanced Flow Correlation Analysis.
Since the DeepCorr dataset was collected using a SOCKS proxy server running Tor clients inside of virtual machines, it is possible that more realistic Tor traffic is somehow different. These changes may lead to less correlated features. We plan to further evaluate DeepCoFFEA using more realistic data settings by employing volunteer Tor clients and varying the client locations to show this impact.

Given that WF attacks have been evaluated in low-data settings, another interesting question is whether more advanced DNN architectures such as ResNet used in Var-CNN [11] can be applied to DeepCoFFEA. With this approach, we could make the flow correlation analysis more scalable by decreasing the size of the training set while yielding comparable performance.

In addition, evaluating DeepCoFFEA using more challenging testing sets collected a couple of years prior to the training set will be an important next step to further decrease

the training complexity of DeepCoFFEA while maintaining good quality of DeepCoFFEA. Finally, further investigation of the DeepCoFFEA architecture for stepping-stone detection and correlation of VPN or HTTPS proxy services might also yield interesting results.

References

[1] Se Eun Oh, Saikrishna Sunkam, and Nicholas Hopper. p1-fp: Extraction, classification, and prediction of website fingerprints with deep learning. *Proceedings on Privacy Enhancing Technologies*, 2019(3):191–209, 2019.

[2] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1928–1943, 2018.

[3] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. In *Network & Distributed System Security Symposium (NDSS)*, 2018.

[4] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. *arXiv preprint arXiv:1708.06376*, 2017.

[5] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. Deepcorr: strong flow correlation attacks on tor using deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1962–1976. ACM, 2018.

[6] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. {RAPTOR}: Routing attacks on privacy in tor. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 271–286, 2015.

[7] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 143–157, 2014.

[8] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. Website fingerprinting at internet scale. In *NDSS*, 2016.

[9] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 1187–1203, 2016.

[10] Payap Sirinam, Nate Mathews, Mohammad Saidur Rahman, and Matthew Wright. Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1131–1148, 2019.

[11] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. Var-CNN: A data-efficient website fingerprinting attack based on deep learning. *Proceedings on Privacy Enhancing Technologies*, 2019(4):292–310, 2019.

[12] Akshaya Mani, T Wilson-Brown, Rob Jansen, Aaron Johnson, and Micah Sherr. Understanding Tor usage with privacy-preserving measurement. In *Internet Measurement Conference*, pages 175–187, 2018.

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

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

[15] Kevin P Dyer, Scott E Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *2012 IEEE symposium on security and privacy*, pages 332–346. IEEE, 2012.

[16] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 227–238, 2014.

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

[18] Tao Wang and Ian Goldberg. Walkie-Talkie: An efficient defense against passive website fingerprinting attacks. In *USENIX Security Symposium*, 2017.

[19] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an Analysis of Onion Routing Security. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 96–114. Springer-Verlag, LNCS 2009, July 2000.

[20] Philippe Boucher, Adam Shostack, and Ian Goldberg. Freedom Systems 2.0 Architecture. White paper, Zero Knowledge Systems, Inc., December 2000.

[21] Jean-François Raymond. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 10–29. Springer-Verlag, LNCS 2009, July 2000.

[22] Xinyuan Wang, Douglas S. Reeves, and S. Felix Wu. Inter-packet delay based correlation for tracing encrypted connections through stepping stones. In *Proceedings of ESORICS 2002*, pages 244–263, October 2002.

[23] Brian N. Levine, Michael K. Reiter, Chenxi Wang, and Matthew K. Wright. Timing attacks in low-latency mix-based systems. In Ari Juels, editor, *Proceedings of Financial Cryptography (FC '04)*, pages 251–265. Springer-Verlag, LNCS 3110, February 2004.

[24] Ye Zhu, Xinwen Fu, Bryan Graham, Riccardo Bettati, and Wei Zhao. On flow correlation attacks and countermeasures in mix networks. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)*, volume 3424 of *LNCS*, pages 207–225, May 2004.

[25] Vitaly Shmatikov and Ming-Hsiu Wang. Timing analysis in low-latency mix networks: Attacks and defenses. In *European Symposium on Research in Computer Security*, pages 18–33. Springer, 2006.

[26] Amir Houmansadr, Negar Kiyavash, and Nikita Borisov. Non-blind watermarking of network flows. *IEEE/ACM Transactions on Networking*, 22(4):1232–1244, 2013.

[27] Xinyuan Wang, Shiping Chen, and Sushil Jajodia. Network flow watermarking attack on low-latency anonymous communication systems. In *2007 IEEE Symposium on Security and Privacy (SP'07)*, pages 116–130. IEEE, 2007.

[28] Amir Houmansadr, Negar Kiyavash, and Nikita Borisov. Rainbow: A robust and invisible non-blind watermark for network flows. In *NDSS*, 2009.

[29] Nick Feamster and Roger Dingledine. Location diversity in anonymity networks. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 66–76. ACM, 2004.

[30] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users get routed: Traffic correlation on tor by realistic adversaries. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 337–348. ACM, 2013.

[31] Rishab Nithyanand, Oleksii Starov, Adva Zair, Phillipa Gill, and Michael Schapira. Measuring and mitigating as-level adversaries against tor. *arXiv preprint arXiv:1505.05173*, 2015.

[32] Matthew Edman and Paul Syverson. As-awareness in tor path selection. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 380–389. ACM, 2009.

[33] Masoud Akhoondi, Curtis Yu, and Harsha V Madhyastha. Lastor: A low-latency as-aware tor client. In *2012 IEEE Symposium on Security and Privacy*, pages 476–490. IEEE, 2012.

[34] Steven J Murdoch and Piotr Zieliński. Sampled traffic analysis by internet-exchange-level adversaries. In *International workshop on privacy enhancing technologies*, pages 167–183. Springer, 2007.

[35] Ryan Wails, Yixin Sun, Aaron Johnson, Mung Chiang, and Prateek Mittal. Tempest: Temporal dynamics in anonymity systems. *Proceedings on Privacy Enhancing Technologies*, 2018(3), June 2018.

[36] Gerry Wan, Aaron Johnson, Ryan Wails, Sameer Wagh, and Prateek Mittal. Guard placement attacks on path selection algorithms for tor. *Proceedings on Privacy Enhancing Technologies*, 2019(4):272–291, 2019.

[37] Axel Arnbak and Sharon Goldberg. Loopholes for circumventing the constitution: Warrantless bulk surveillance on americans by collecting network traffic abroad, 2014.

[38] Joshua Juen, Aaron Johnson, Anupam Das, Nikita Borisov, and Matthew Caesar. Defending tor from network adversaries: A case study of network path prediction. *Proceedings on Privacy Enhancing Technologies*, 2015(2):171–187, 2015.

[39] Henry Tan, Micah Sherr, and Wenchao Zhou. Data-plane defenses against routing attacks on tor. *Proceedings on Privacy Enhancing Technologies*, 2016(4):276–293, 2016.

[40] Armon Barton and Matthew Wright. Denasa: Destination-naive as-awareness in anonymous communications. *Proceedings on Privacy Enhancing Technologies*, 2016(4):356–372, 2016.

[41] Yixin Sun, Anne Edmundson, Nick Feamster, Mung Chiang, and Prateek Mittal. Counter-raptor: Safeguarding tor against active routing attacks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 977–992. IEEE, 2017.

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

[43] Chih Chang and Chih Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[44] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[45] M.W Gardner and S.R Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment*, 32(14-15):2627–2636, aug 1998.

[46] S. Lawrence, C.L. Giles, Ah Chung Tsoi, and A.D. Back. Face recognition: a convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, 1997.

[47] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 37–49, 2012.

[48] G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786), 2006.

[49] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[50] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[51] Fereshteh Falah Chamasemani and Yashwant Prasad Singh. Multi-class Support Vector Machine (SVM) Classifiers – An Application in Hypothyroid Detection and

Classification. In *2011 Sixth International Conference on Bio-Inspired Computing: Theories and Applications*, pages 351–356. IEEE, Sep 2011.

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

[53] Tao Wang and Ian Goldberg. Improved website fingerprinting on tor. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 201–212, 2013.

[54] David Wagner, Bruce Schneier, et al. Analysis of the ssl 3.0 protocol. In *The Second USENIX Workshop on Electronic Commerce Proceedings*, pages 29–40, 1996.

[55] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *Proceedings - IEEE Symposium on Security and Privacy*, pages 191–206, 2010.

[56] Alexander Schaub, Emmanuel Schneider, Alexandros Hollender, Vinicius Calasans, Laurent Jolie, Robin Touillon, Annelie Heuser, Sylvain Guilley, and Olivier Rioul. Attacking Suggest Boxes in Web Applications Over HTTPS Using Side-Channel Stochastic Algorithms. *Risks and Security of Internet and Systems*, 2015.

[57] Sampreet A. Sharma and Bernard L. Menezes. Implementing side-channel attacks on suggest boxes in web applications. In *Proceedings of the First International Conference on Security of Internet of Things - SecurIT '12*, pages 57–62, New York, New York, USA, 2012. ACM Press.

[58] K Zhang, Z Li, R Wang, X F Wang, and S Chen. Sidebuster: Automated detection and quantification of side-channel leaks in web application development. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 595–606, 2010.

[59] Peter Chapman and David Evans. Automated Black-Box Detection of Side-Channel Vulnerabilities in Web Applications. *Proceedings of the 18th ACM conference on Computer and communications security - CCS '11*, (October):263, 2011.

[60] M Backes, G Doychev, and B Köpf. Preventing Side-Channel Leaks in Web Traffic: A Formal Approach. *NDSS*, 2013.

[61] Daniel C. Howe and Helen Nissenbaum. TrackMeNot: Resisting Surveillance in Web Search. *Lessons from the Identity Trail: Anonymity, Privacy and Identity in a Networked Society*, pages 417–436, 2009.

[62] Josep Domingo - Ferrer, Agusti Solanas, and Jordi Castella - Roca. $h(k)-$private information retrieval from privacy-uncooperative queryable databases. *Online Information Review*, 33(4):720–744, Aug 2009.

[63] Ero Balsa, Carmela Troncoso, and Claudia Diaz. OB-PWS: Obfuscation-based private web search. In *Proceedings - IEEE Symposium on Security and Privacy*, pages 491–505, 2012.

[64] Marc Juarez and Vicenc Torra. DisPA: An Intelligent Agent for Private Web Search. pages 389–405. Springer International Publishing, 2015.

[65] Matthew Fredrikson and Benjamin Livshits. RePriv: Re-imagining Content Personalization and In-browser Privacy. In *2011 IEEE Symposium on Security and Privacy*, pages 131–146. IEEE, May 2011.

[66] Tor-Browser-Crawler. https://github.com/webfp/tor-browser-crawler.

[67] Keyword-Tool. http://keywordtool.io.

[68] Gregory Dudek. Aol-user-ct-collection. http://www.cim.mcgill.ca/~dudek/206/Logs/AOL-user-ct-collection//.

[69] Google-Instance-Disliked-Blacklist-Words. https://www.2600.com/googleblacklist/.

[70] Internet live stats. http://www.internetlivestats.com/one-second/#google-band.

[71] Tor project. https://www.torproject.org/.

[72] tshark. https://www.wireshark.org/docs/man-pages/tshark.html.

[73] William H Kruskal and W Allen Wallis. Use of Ranks in One-Criterion Variance Analysis. *Source Journal of the American Statistical Association*, 4710087:583–621, 1952.

[74] Google's people also ask. http: //www.internetmarketingninjas.com/blog/search-engine-optimization/googles-people-also-ask-related-questions/.

[75] Google launches knowledge graph, 'first step in next generation search'. https://searchenginewatch.com/sew/news/2175783/google-launches-knowledge-graph-step-generation-search.

[76] Google adds "people also search for" thumbnails to search results. http://www.thesempost.com/google-adds-people-also-search-for-thumbnails-to-search-results/.

[77] Tao Wang and Ian Goldberg. On Realistically Attacking Tor with Website Fingerprinting. *Proceedings on Privacy Enhancing Technologies*, (4):21–36, 2016.

[78] Brooke McDonald. How often does google update its search results? https://hdwebpros.com/blog/how-often-does-google-update-its-search-results.html, 2013.

[79] Tao Wang and Ian Goldberg. Walkie-talkie: An efficient defense against passive website fingerprinting attacks. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 1375–1390, 2017.

[80] Se Eun Oh, Shuai Li, and Nicholas Hopper. Fingerprinting keywords in search queries over tor. *Proceedings on Privacy Enhancing Technologies*, 2017(4):251–270, 2017.

[81] Rebekah Overdorf, Mark Juarez, Gunes Acar, Rachel Greenstadt, and Claudia Diaz. How unique is your. onion? an analysis of the fingerprintability of tor onion services. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2021–2036, 2017.

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

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

[84] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1097–1105, 2012.

[85] Alan E Robinson, Paul S Hammon, and Virginia R de Sa. Explaining brightness illusions using spatial filtering and local response normalization. *Vision Research*, 47(12):1631–1644, 2007.

[86] Tensorflow. https://www.tensorflow.org/.

[87] Tflearn. http://tflearn.org/.

[88] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *USENIX Security Symposium*, pages 1187–1203, 2016.

[89] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International Conference on Machine Learning*, pages 115–123, 2013.

[90] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123, 2013.

[91] Tor browser crawler. https://github.com/webfp/tor-browser-crawler.

[92] Milad Nasr, Amir Houmansadr, and Arya Mazumdar. Compressive traffic analysis: A new paradigm for scalable traffic analysis. In *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2017.

[93] Amir Houmansadr, Negar Kiyavash, and Nikita Borisov. Non-blind watermarking of network flows. *IEEE/ACM Transactions on Networking*, 22(4):1232–1244, 2014.

[94] Ali Mousavi, Ankit B Patel, and Richard G Baraniuk. A deep learning approach to structured signal recovery. In *Communication, Control, and Computing (Allerton), 2015 53rd Annual Allerton Conference on*, pages 1336–1343. IEEE, 2015.

[95] Christos Louizos, Kevin Swersky, Yujia Li, Max Welling, and Richard Zemel. The variational fair autoencoder. *arXiv preprint arXiv:1511.00830*, 2015.

[96] Random forests. *Machine Learning*, 45(1):5–32, 2001, /dx.doi.org/10.1023%2FA%3A1010933404324.

[97] Mohammad Saidur Rahman, Payap Sirinam, Nate Mathews, Kantha Girish Gangadhara, and Matthew Wright. Tik-Tok: The utility of packet timing in website fingerprinting attacks. *Proceedings on Privacy Enhancing Technologies*, 2020(3):5–24, 2020.

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

[99] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[100] Bruno Lecouat, Chuan-Sheng Foo, Houssam Zenati, and Vijay R Chandrasekhar. Semi-supervised learning with gans: Revisiting manifold regularization. *arXiv preprint arXiv:1805.08957*, 2018.

[101] Se Eun Oh, Shuai Li, and Nicholas Hopper. Fingerprinting keywords in search queries over Tor. *Proceedings on Privacy Enhancing Technologies*, 2017(4):171–190.

[102] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In

Advances in Neural Information Processing Systems (NeurIPS), pages 2672–2680, 2014.

[103] Does Alexa have a list of its top-ranked websites ? – Alexa support. https://support.alexa.com/hc/en-us/articles/200449834-Does-Alexa-have-a-list-of-its-top-ranked-websites-.

[104] Jost Tobias Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. *arXiv preprint arXiv:1511.06390*, 2015.

[105] Ilya Sutskever, Rafal Jozefowicz, Karol Gregor, Danilo Rezende, Tim Lillicrap, and Oriol Vinyals. Towards principled unsupervised learning. *arXiv preprint arXiv:1511.06440*, 2015.

[106] Code for the paper "improved techniques for training GANs. https://github.com/openai/improved-gan.

[107] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[108] Rob Jansen, Marc Juarez, Rafa Galvez, Tariq Elahi, and Claudia Diaz. Inside job: Applying traffic analysis to measure Tor from within. In *Network & Distributed System Security Symposium (NDSS)*, 2018.

[109] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 901–909, 2016.

[110] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6626–6637, 2017.

[111] Paul Syverson, R Dingledine, and N Mathewson. Tor: The second generation onion router. In *Usenix Security*, 2004.

[112] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[113] K Krishna and M Narasimha Murty. Genetic k-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 29(3):433–439, 1999.

[114] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[115] Keras: The python deep learning library. https://keras.io/.

[116] Tensorflow. https://www.tensorflow.org/.

[117] Jiajun Gong and Tao Wang. Zero-delay lightweight defenses against website fingerprinting. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pages 717–734, 2020.

[118] James K Holland and Nicholas Hopper. Regulator: A powerful website fingerprinting defense. *arXiv preprint arXiv:2012.06609*, 2020.

[119] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[120] Weidi Xu, Haoze Sun, Chao Deng, and Ying Tan. Variational autoencoder for semi-supervised text classification. In *AAAI Conference on Artificial Intelligence*, 2017.