DFD: Adversarial Learning-based Approach to Defend Against Website Fingerprinting

Ahmed Abusnaina[†]
University of Central Florida
Orlando, FL, USA
ahmed.abusnaina@knights.ucf.edu

Rhongho Jang[†]
University of Central Florida
(UCF) & Inha University
r.h.jang@knights.ucf.edu

Aminollah Khormali
University of Central Florida
Orlando, FL, USA
aminkhormali@knights.ucf.edu

DaeHun Nyang
Inha University
Incheon, South Korea
nyang@inha.ac.kr

David Mohaisen
University of Central Florida
Orlando, FL, USA
mohaisen@ucf.edu

***Abstract*—The Onion Router (Tor) is designed to support an anonymous communication through end-to-end encryption. To prevent vulnerability of side channel attacks (*e.g.* website fingerprinting), dummy packet injection modules have been embedded in Tor to conceal trace patterns that are associated with the individual websites. However, recent study shows that current Website Fingerprinting (WF) defenses still generate patterns that may be captured and recognized by the deep learning technology. In this paper, we conduct in-depth analyses of two state-of-the-art WF defense approaches. Then, based on our new observations and insights, we propose a novel defense mechanism using a per-burst injection technique, called Deep Fingerprinting Defender (DFD), against deep learning-based WF attacks. The DFD has two operation modes, one-way and two-way injection. DFD is designed to break the inherent patterns preserved in Tor user's traces by carefully injecting dummy packets within every burst. We conducted extensive experiments to evaluate the performance of DFD over both closed-world and open-world settings. Our results demonstrate that these two configurations can successfully break the Tor network traffic pattern and achieve a high evasion rate of 86.02% over one-way client-side injection rate of 100%, a promising improvement in comparison with state-of-the-art adversarial trace's evasion rate of 60%. Moreover, DFD outperforms the state-of-the-art alternatives by requiring lower bandwidth overhead; 14.26% using client-side injection.**

I. Introduction

The Onion Router (Tor) provides anonymous communication to more than two million daily Internet users [1] to hide their location and online activity, *e.g.* website visits, instant messages, posts, etc. from those who conduct traffic analysis and network monitoring, such as Internet Service Providers (ISPs). To set out, first the communications' content and routing information is encrypted, then the encrypted traffic is relayed through a route that is built by random selection of nodes among more than seven thousand nodes [2], such that only a single node knows its immediate peers but never the origin and destination of a communication at the same time.

Previous research works have investigated the privacy aspects of Tor network, showing that Tor network is prone to side-channel attacks where a local adversary can infer which website was visited by a user [3]. The adversary can access communication's metadata from the side-channel leakage, for instance, using the side channel. Moreover, the adversary may have access to the direction and size of the encrypted network packets. Such information can be utilized to construct a unique fingerprint, allowing network eavesdroppers to reveal which website was visited based on the network traffic.

Particularly, Website Fingerprinting (WF) is a traffic analysis attack that allows an attacker to recognize the patterns of visited websites, exploiting the content differences, in a connection that is encrypted using Tor software. A large body of research has investigated the feasibility of WF in breaking the privacy that is offered by Tor [4], [5], [6], [7], [8]. WF can be considered as a pattern recognition problem from a machine learning point of view. Generally, the attacker first trains a classifier over a set of representative traffic features extracted from a large number of websites, then uses this model to predict a victim's traces as one of those websites. Therefore, several research works attempted to improve the effectiveness of WF by introducing various sets of hand-crafted features that represent Tor traffic and state-of-the-art machine learning algorithms [5], [9], [10]. Although machine learning-based classifiers are able to achieve more than 91% classification accuracy [10], [11], [5], their performance is highly dependent not only on the structure of the classifier but also on the manually extracted traffic features. In addition, those features are not robust against changes in the network protocols [12], [13], [8]. A few research works attempted to address these issues by using deep learning [7], [8]. Deep learning has shown to outperform traditional machine learning networks and does not need feature engineering.

Machine learning networks are widely used in a wide range of applications [14], [15], [16], [17], [18], [19]. Despite the unique characteristics of deep learning-based models, it has been shown that they are vulnerable to Adversarial Examples (AEs) [20]. AEs are carefully crafted samples by applying small perturbation to the input dataset, leading the classifier

[†]These authors contributed equally.

to misclassification. We note that AEs are similar to the original inputs, and not necessarily outside of the training data manifold [21]. Adversarial learning is an active research area and several algorithms for generating adversarial examples are presented, such as the fast gradient sign method [22]. The primary goal of these adversarial attack methods is to generate AEs such that not only reduces the confidence of the classifier, but also forces the model to generate adversary's desired output. Although adversarial machine learning has been an active research area, mostly to compromise the performance of the machine and deep learning models in sensitive domains, there are only very few research works that investigate their application as a defensive strategy against website fingerprinting [23]. Such defense strategies need to be investigated in more detail, *e.g.* evasion rate, practicality, and bandwidth overhead.

Goal of this study. Motivated by the aforementioned research gap, the main goal of this study is to improve the privacy of the users of Tor network against WF attacks by introducing a novel defense mechanism based on adversarial learning technique.

Approach. To tackle the above objectives, first we conducted fully automated deep learning-based WF attack that works based on raw traffic traces and does not require hand-crafted features. It should be noted that deep learning-based WF attacks are designed to be more resistant to changes in the features introduced by defenses. Second, we investigate the performance of two well-established defense methods, including WTF-PAD [13] and Walkie-Talkie [24]. Finally, we explore the feasibility of adversarial learning as a potential defense strategy against deep learning-based WF attacks, while preserving the practicality of the traffic traces. To set out, we looked into the defense strategy and real-world traces of WTF-PAD and W-T. Through analysis, we observed a potential flaw that leads to residual patterns to be captured by the deep learning models. Based on our new findings, we proposed a novel approach, Deep Fingerprinting Defender (DFD), which maintains the practicality of the generated adversarial traffic traces while achieving a high misclassification rate; which maintains Tor users' privacy.

Contributions. In this paper, we are making the following contributions:

❶ We conduct in-depth analyses of the working flow of WTF-PAD and W-T. Particularly, we looked into the strategies of both approaches to find potential behaviors that can expose features to the deep learning-based attacks. We described our observations and explained rationals of these attack surfaces.

❷ To address the issues associated with previous defense schemes, we proposed DFD, per burst injection technique to defend against deep learning-based website fingerprinting.

❸ Through extensive experiments, it has been shown that DFD can dramatically reduce the success rate of the adversary in identifying user's visiting website. DFD shows a good misclassification performance in both open-world and close-world settings. Further, we evaluate the performance of DFD against deep learning models trained on its generated traces, proposing an optimization to overcome traces pattern recognition.

Organization. This work is organized as follows: The related work is reviewed in section II. In section III, a threat model and adversarial settings are provided. Traditional WF defenses and DFD system designs are described in section IV. The evaluation of the proposed approach is in section V. Finally, the work is concluded in section VI.

II. Related Work

WF attacks. WF attacks against Tor was first evaluated by Herrmann *et al.* in 2009 [25]; their classification accuracy was only 3% in a closed world of 775 websites. However, later in 2011, Panchenko *et al.* were able to achieve a classification accuracy of 55% on the same dataset using an improved set of engineered features [26]. Moreover, the success rate of WF attacks improved gradually up to 90% using classifiers that perform based on edit-distances [27], [28]. However, these classifiers were not practical due to their high computational costs. Recently, researchers utilized both more advanced machine along with sophisticated features set that not only improved the attack success rate but also could be deployed in real-world due to their reduced attack costs. Wang *et al.* [11] presented a WF attack based on the k-Nearest Neighbors (k-NN) classifier that measures the similarity between websites based on a set of features, such as the number of incoming and outgoing cells, the numbers of bursts, packet ordering, etc.. This approach was able to achieve an accuracy rate of 91% in a closed world of 100 websites. Moreover, Panchenko *et al.* [5] presented WF attack based on Support Vector Machine (SVM) classifier that performs over a set of engineered features constructed using the cumulative sum of packet lengths. Similarly, they achieved a success rate of 91% in a closed-world setting. Hayes and Danezis [10] presented k-fingerprinting, an elegant WF attack method using random forests.

DL-based WF Attacks. In order to overcome the shortcomings of traditional machine learning-based WF attacks, researchers presented deep learning-based WF attacks. For instance, Abe and Goto [29] introduced the application of Stacked Denoising Autoencoders (SDAE) to WF attacks. Moreover, CNN-based website fingerprinting is presented by Lu *et al.* [30]. Similarly, Rimmer *et al.* [8] conducted a systematic study to incorporate deep learning networks into WF. The proposed method is able to reach a success rate of 96% and 94% on closed-world of 100 websites and 900 websites, respectively. Moreover, Sirinam *et al.* [7] presented a deep learning-based WF attack against Tor. Their approach is able to achieve more than 98% success rate on *undefended* Tor traffic, outperforming existing WF attacks. In addition, for *defended* Tor traffic it was able to reach 90% and 49.7% success rate on WTF-PAD and Walkie-Talkie, respectively.

WF defenses. In general, WF attack methods aim to increase the misclassification rate of the adversary through different strategies, such as injecting dummy packets, delaying packets, etc. For example, Dyer *et al.* [31] introduced BuFLO, a traffic modification mechanism that removes packets' specific features, thus making the traffic look constant. Later, Cai *et* *al.* [32], [33] attempted to improve the function of BuFLO by grouping sites based on their size. Then all sites in a group are padded to the largest size in that group. However, these methods were expensive in terms of bandwidth and latency overheads. To overcome these issues, Juarez *et al.* [13] proposed Website Traffic Fingerprinting Protection with Adaptive Defense (WTF-PAD). WTF-PAD is lightweight and causes small bandwidth overhead while incurring zero-latency overhead. Wang and Goldberg [24] proposed Walkie-Talkie, a solution that works based on half-duplex communication methods. Imani *et al.* [23] introduced *adversarial traces* based on padding techniques, and add those traces to the Tor traffic as a defense against WF attacks. Their approach drooped the accuracy of deep learning based WF attack from 98% to 60%, while resulting in 47% bandwidth overhead.

Adversarial Methods. Machine/deep learning networks are actively used in many sensitive applications and domains, including website fingerprinting [7], [8]. However, it has been shown that adversaries can manipulate their output using adversarial examples [34]. Note that such an attack can be utilized to implement a defense strategy against website fingerprinting, and to undermine the adversary's ability to identify the Tor users' visited websites [23]. Recently, different adversarial attack algorithms have been introduced by the research community. For example, Goodfellow *et al.* [22] introduced FGSM, image-based adversarial method that forces the model to misclassification through crafted AEs. Further, Abusnaina *et al.* [35] introduced the adversarial examples into the field of anomaly detection. In this work, we leverage the idea of applying perturbation into implementing a defense against website fingerprinting.

III. Threat Model

Despite Tor's goal in protecting users' privacy, adversarial entities are able to undermine Tor's protection and identify users activity profiles through traffic analysis techniques. There are several research works that have demonstrated adversary's capability in exploiting network traffic patterns to identify visited pages by a Tor user [10], [5], [6], [7], [8]. To set out, first, the adversaries monitor traffic sequences of their own visits to a set of known websites, including websites they are interested in detecting. Then, a set of representative features, such as packet size [25], time and volume [36], edit-distance score [28], rate of traffic bursts in both directions [9], etc. are extracted from the captured traffic flows. Finally, the extracted features can be utilized to train a machine/deep learning classifier that predicts Tor users' visited websites.

In this study, we considered the threat model presented in [7], [8], as shown in Figure 1, where the adversary has access only to the link between the entry node of the Tor network and the user, through which he can only monitor network packets in a passive manner. A passive adversary is only able to record the transmitted network packets during the communication, however, the adversary is not able to change, delay, drop, or insert new packets to the sequence of packets. All entities that have such level of access to the network traffic can conduct WF attacks, such as Autonomous Systems (AS), ISPs, local network administrators, etc. In this work, an ISP-level adversary launches the WF attack through collecting traffic at the Transmission Control Protocol (TCP) layer. Note that it is generally assumed that the encryption provided by Tor network cannot be decrypted by the adversary.

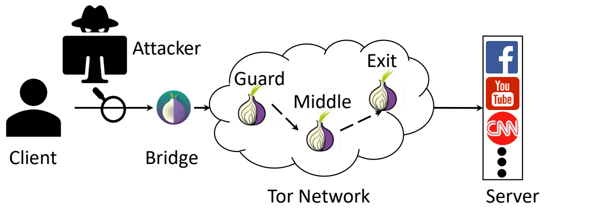


Figure 1. Website Fingerprinting Threat Model. The adversary observes the traffic between the client and the entry point of the Tor network.

Closed/Open-world Setting. WF attacks are evaluated in two scenarios, a closed-world and an open-world of websites. The closed-word assumes that users can only visit a small set of websites and that the adversary has samples to train his models on all of them [3]. This assumption was criticized for being unrealistic [4], [37], as the population of sites that can be potentially visited is so large that not even the most powerful adversaries have the resources to collect data and train for every site. Subsequent studies have considered an open-world scenario, a more realistic setting in which the adversary can only train on a small fraction of the sites the user can visit. We use the closed-world experiments for a detailed comparison of different algorithms and parameter settings, and we report the results of the open-world experiments for a more realistic evaluation of the attack.

Adversarial Settings. The main goal of the adversary in adversarial machine learning is to fool The deep learning classifier f using adversarial examples x', generated by applying small perturbations δ into input sample x. The adversary uses the crafted adversarial example to force the model to generate his desired output, *e.g.* misclassification. Attacks on deep learning networks can be categorized from different points of view, including adversary's goals and capabilities [34]. The followings are the description of each point of view.

- **Adversarial Goals.** The main goal of the adversary in deep learning systems is to force the model into incorrect results. The main goal of the adversary can be categorized based on the nature of the incorrectness into confidence reduction, untargeted misclassification, and targeted misclassification.
- **Adversarial Capabilities.** Considering attacks conducted at the test time, the adversarial attacks are divided into white-box attacks and black-box attacks. While in white-box attacks the adversary has access to either of the model architecture or the training data, in black-box attacks the adversary has only oracle access to the model. In this study, we initially assume an adversary with no prior knowledge of the classifier and only has an oracle access to the model. Later, we loosen up the assumption, where

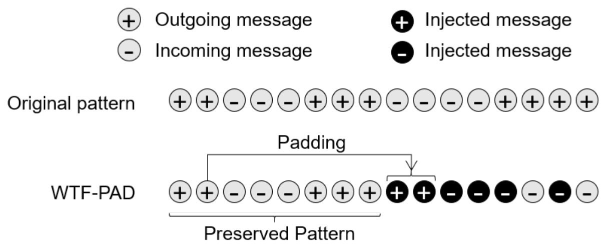


Figure 2. WTF-PAD: message padding after each burst.

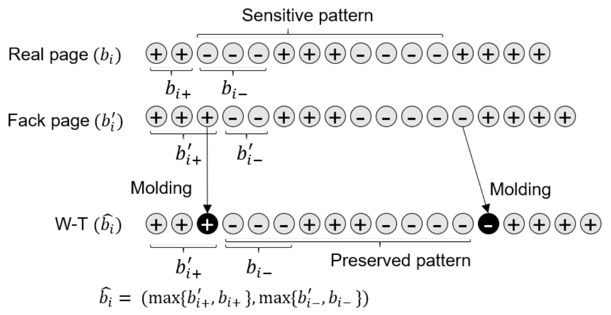


Figure 3. W-T: burst molding using the burst sequence of the real page.

the adversary is aware of the deep learning classifier and the used defense. In addition, the adversary's goal is to conduct an untargeted misclassification.

Deep Learning-based Attacks. Deep learning has been utilized to produce a state-of-the-art classification accuracy in several research fields. In our work, we mainly focus on two well-established deep learning networks to conduct WF, the Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs). **DNN** is a type of feed-forward artificial neural network with multiple hidden layers between the input and the output layers [38]. DNN focuses on turning the input into the output by manipulating the weights of the links and calculating the probability of each output. Similarly, **CNN** is a convolutional layer-based classifier, usually used to extract deep patterns within sequences. Both CNNs and DNNs are well-known for their unique characteristics in automatic feature extraction, dealing with a large number of features, providing high performance, and requiring minimal pre-processing effort [39]. Recently, researchers have employed deep learning algorithms to identify Tor users' browsing patterns by abusing their network traffic. Therefore, they are suitable candidates for conducting WF attacks using network traffic traces [30].

IV. Methodology

A. Traditional WF Defenses

The main focus of website fingerprinting (WF) defense is to decrease the adversary's success rate in predicting the Tor users' visited websites. Recently, two different defense techniques have been deployed in Tor network, namely, Website Traffic Fingerprinting Protection with Adaptive Defense (WTF-PAD) [13] and Walkie-Talkie (W-T) [24]. Both of those techniques are based on the dummy message injection, but employed in different ways. Although these defense strategies are well-appreciated by the Tor community due to their low bandwidth and latency overheads, a recent study has shown that they are vulnerable to deep learning-based WF attacks [7]. In the following, we describe these defense schemes and provide rational explanations as to why they are vulnerable to DL-based attack, which was missing in the original paper [7].

WTF-PAD. WTF-PAD was proposed by Juarez *et al.* [13] as a defense method against website fingerprinting. This defense is based on Adaptive Padding that is originally designed to defend against the end-to-end traffic analysis [40]. The term "adaptive" here means that the defender injects dummy messages following the real-world distribution of inter-packet arrival time to disrupt the time feature from the packet flow. Meanwhile, by injecting these dummy messages at the end of each burst (*i.e.,* burst padding), the volume information and burst boundary are concealed for defending against WF. While WTF-PAD was shown to be robust with traditional machine learning (*i.e.,* k-NN, Pa-SVM [26], DL-SVM [28], VNG++ [31] etc.), it is vulnerable with CNN and DNN [7].

Observation. Figure 2 shows an example of the original and WTF-PAD traces from our dataset. Basically, WTF-PAD is a zero-delay scheme, which means that the real messages are sent out without buffering or blocking. Thanks to this design, WTF-PAD provides a good user experience to users surfing sensitive web pages. However, this design also exposes the traffic features and patterns to adversaries. As shown in Figure 2, and since WTF-PAD starts the padding process once the idle time of the egress port expires with a certain threshold, the real messages can go back and forth before the padding. More specifically, the full-duplex communication mechanism of WTF-PAD results in a small inter-burst time, which means two continuous real bursts can be observed in the packet flow. Moreover, the receiving of TAGs (*e.g. img*) causes immediately a request of the corresponding resources, which can lead to three continuous real bursts combining the previous example. To this end, these short patterns are preserved and repeated in the packet flows, which is a strong feature combination exposed to the adversaries. In this paper, we reproduced the DL model proposed by Sirinam *et al.* [7] and achieved similar detection accuracy as reported.

W-T. W-T is based on half-duplex communication with the web server [24], where both the client and the proxy maintain a buffer. The buffer is mainly responsible for queuing and sending out messages as a burst without intersecting with the burst from the peer. W-T conceals the time features by sending each message with a constant time interval. Meanwhile, and for hiding the sensitive message sequence information of the real page, it simulates the connection of a fake page then mixes the two burst patterns in what is known as "molding".

Observation. Figure 3 shows an example of the original and W-T traces from our dataset. Let's assume the sequence of burst of the real page is $b_i = (b_{i+}, b_{i-})$, where b_{i+} stands for the outgoing bursts and b_{i-} is the incoming bursts. By simulating the fake page, we have the fake sequence of burst $b'_i = (b'_{i+}, b'_{i-})$. As shown in Figure 3, W-T compares the length of each i-th burst and sends the real burst with the length of the longer burst, namely the burst molding (*i.e.,* $\hat{b_i} = (max\{b_{i+}, b'_{i+}\}, max\{b_{i-}, b'_{i-}\})$). However, the design of W-T leaves an opportunity for adversaries to capture the features. Since the bursts of the fake page can either be longer or shorter than the real page, 0% of molding happens in the worst case and 100% in the best case. Overall, the quality of the selected fake page determines how many sensitive burst sequences can be hidden. However, it is difficult to determine the fake page because the burst sequence of the real page is unpredictable. We also repeated DL-based WF attack on W-T and achieved a near 50% accuracy, which is the maximum according to the theory of W-T [24] (See subsection V-C).

Summary. Through the discussions above, we can conclude that neither the burst padding nor the random hiding of the burst is helpful in defending against the DL-based WF attack since the real burst sequences can be preserved and captured by The deep learning models (*i.e.,* CNN and DNN). Through these observations, however, we can infer that injecting dummy messages within a burst is a more efficient way to break the patterns of the real trace than injecting them after the burst, as injecting packets within the burst is more likely to conceal the patterns. Based on these findings, we designed a burst injection-based WF defense, which aims to hide the burst sequences. Our design is based on an experiment-driven approach, by simulating different injection scenarios.

B. Deep Fingerprinting Defender

Our proposed defense scheme, DFD, is a client-side dummy message injection solution that aims to conceal the sequence pattern within the packet flow and to provide a low bandwidth overhead as well. DFD is designed to inject dummy messages into every outgoing burst for hiding the real burst pattern. At the same time, the amount of injection follows a targeted manner (*i.e.,* perturbation rate p) so that the overall bandwidth overhead can be limited with with a fixed boundary. In this paper, we do not consider the time features (*i.e.,* inter-packet arrival time and inter-burst time) since they are insignificant in WF attacks [24], [7]. The structure of the DFD is shown in Figure 4. As demonstrated in Figure 4, DFD is composed of two modules, the *Burst observer* and the *Injection buffer*.

ⓐ **Burst Observer.** The main purpose of this module is monitoring bursts of outgoing messages. In this paper, we refer the burst as a set of continuous messages such that the length is larger than 2. Once the DFD start running, *Burst observer* records the length of the burst (l_i) and triggers the injection event when the new burst arrives. For determining the number of dummy messages to inject within the newly arriving burst, DFD takes perturbation rate (P) as a parameter then multiply it with the length of each burst (*i.e.,* $P \times l_i$). By doing so, the overhead of injection is P of the total amount of the outgoing messages. However, since the length of the ongoing burst is unpredictable, we approximate the overhead by using the length of the last recorded burst (*i.e.,*

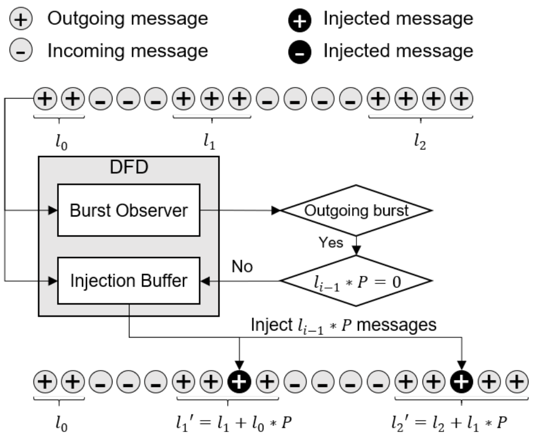


Figure 4. General structure of DFD approach. Here l_i refers to the length of each outgoing burst and P is the perturbation rate. Note that $P = 0.5$ in this example.

$P \times l_{i-1}$). Therefore, the total amount of injection (I_{total}) is $I_{total} = \sum_{i=0}^{n-1} P \times l_{i-1} \approx \sum_{i=0}^{n-1} P \times l_i$, where n is the total number of bursts. By doing so, the total injection overhead can be limited by around P (percent) of the outgoing messages.

ⓑ **Injection Buffer.** This module is responsible for injecting the dummy messages into the current outgoing burst. For minimizing the injection delay, we duplicate and buffer the passed real messages by maintaining an injection buffer. As such, we could inject dummy messages within the outgoing burst in a timely manner. To avoid the additional bandwidth overhead, we use the previous acknowledged messages for injection. Note that adding dummy messages to the packet flow will not affect the practicality of the generated adversarial packet flows. HTTP/HTTPS are based on the TCP transmission protocol, where each TCP packet has a sequence number, once it is received by the server, an acknowledgement (ACK) is sent to the client to inform of the packet arrival. If the server receives a previously acknowledged packet, the server will simply discard it [41], [42]. In DFD, the dummy message is selected from the previous ACKs, to ensure that it will be discarded by the server without affecting the communication.

Algorithm 1 shows the steps taken by *Burst observer* to signal the *Inject Buffer*. *Burst Observer* takes the packet flow f and the perturbation rate P as inputs. For every outgoing message (msg) from f, *Burst Observer* counts the length of the ongoing burst (*i.e.,* continuously outgoing messages). The counting of bursts is interrupted by the incoming message indicating the end of the burst. As a result, the length of the burst is stored in l_{i-1} and starts the new round counting. To this end, l_{i-1} is continuously updated with the length of the last burst (lines 9-12). The injection event is triggered during the ongoing burst (line 7-8). Once the length of continuous messages exceeds two, which indicates the outgoing messages become a burst, it signals *Inject Buffer* to inject $l_{i-1} \times P$ messages into the ongoing burst, where l_{i-1} is the length of the last burst and P is the perturbation rate. By doing so, the overhead caused by the injection can be limited by P of

Algorithm 1: Burst observer module algorithm

```
1  Function BurstObserver ($f$, $P$);
   Input : Packet flow $f$, Perturbation rate $P$
2  $Burst_{Count} \leftarrow 0$
3  $l_{i-1} \leftarrow 0$
4  foreach $msg$ in $f$ do
5      if (msg is outgoing) then
6          $Burst_{Count}++$
7          if ($Burst_{Count}$ = 2) then
8              Signal InjectorBuffer($P \times l_{i-1}$)
9          end
10     else if (msg is incoming) then
11         if ($Burst_{Count} \neq 0$) then
12             $l_{i-1} \leftarrow Burst_{Count}$
13             $Burst_{Count} \leftarrow 0$
14         end
15     end
16 end
```

the outgoing messages. *Inject Buffer* is a non-block function that only captures the injection signal from *Burst Observer*. It maintains an injection buffer that duplicates and stores the previous outgoing ACK message for injecting messages without delay. Note that the previous ACK will be ignored by the server due to the TCP protocol operation.

V. Experiments and Evaluation

A. Dataset

To evaluate the efficiency of WF defense techniques, we obtained the WF datasets collected by Sirinam *et al.* [7]. A brief description of the utilized datasets is in the following.

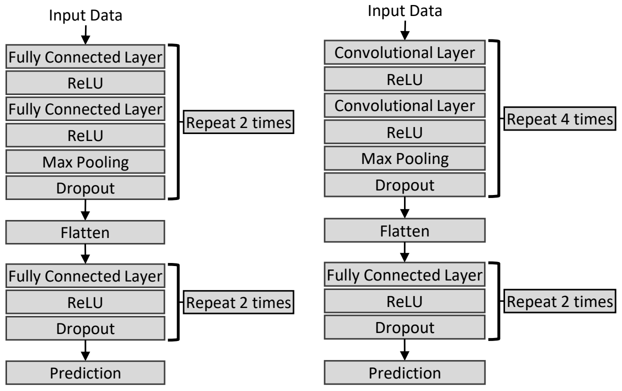
Closed-World Setting (CWS). Sirinam *et al.* [7] collected fingerprints of websites by visiting the homepage of each of the top Alexa 100 sites for 1250 times using tor-browser-crawler [4]. Corrupted and short traffic traces are discarded in a pre-processing phase. The final dataset contained 1000 visit traces of 95 known websites.

Open-World Setting (OWS). For an open-world dataset, Sirinam *et al.* visited the top Alexa 50,000 sites excluding the first 100 websites used in the closed-world dataset. They obtained a trace of each visited website, then they filtered the corrupted traces, along with access denied and blank pages responses. The final open-world dataset consists of 40,716 unmonitored traffic traces along with the 95,000 monitored traffic traces from the closed-world dataset.

Data Representation. Each visit of a website is considered as a sample, where each sample is represented by a vector of size 1×5000 indicating the direction of the packets (-1 for incoming and $+1$ for outgoing). For training purposes, each website is identified with a unique integer number. The label of a sample is the corresponding website identifier. In the open-world, all the samples of unmonitored websites were labeled with a single identifier.

B. Implementation

This section provides details into our system implementation, including The deep learning based WF attacks and DFD.



(a) DNN-based WF attack (b) CNN-based WF attack

Figure 5. The general architecture of the designed DL-based WF attacks.

1) DL-based WF Implementation: We evaluated the performance of the DFD over two deep learning-based WF attacks. These models are trained over a large closed-world dataset, described in subsection V-A. In the following, we provide a brief description of the implementation of these WF attacks.

DNN-based WF Attack. Based on the characteristics of DNNs, i.e., extracting representative features automatically, we designed a DNN-based network to identify the Tor users' visited websites. The general architecture is shown in 5(a). The model is composed of two main blocks. The first block contains two fully connected layers with ReLU activation function, followed by max pooling and dropout layers. The second block consists of one fully connected layer with ReLU activation, followed by a dropout layer. We set the number of epochs to 50, a batch size of 256, and a filter size of 64.

CNN-based WF Attack. The general structure of our design for the CNN model is shown in 5(b). The implemented CNN consists of two main blocks: convolutional layers and fully connected layers. The first block consists of two consecutive convolutional layers with ReLU activation function, followed by max pooling and dropout layers. The second block consists of a fully connected layer with ReLU activation function, followed by a dropout layer. We set the number of epochs to 50, with a batch size of 256 and a filter size of 64.

2) DFD Implementation: To evaluate our DFD, we developed two operation modes for different injection scenarios, namely one-way injection and two-way injection.

One-way Injection (OWI). In one-way injection, dummy message injection can be performed by either the client or the server-side. For website surfing, the client-side injection results in less bandwidth overhead and a reasonable misclassification rate. Moreover, the implementation of one-way injection can be done without modifying the server-side codes.

Two-way Injection (TWI). In two-way injection, we use the symmetric and asymmetric injections. The purpose of the two-way injection is for a higher misclassification rate at the cost of increasing the bandwidth overhead. For our implementation, we need to modify both the client and the server code. For simplicity, and instead of modifying the web server, we can

Table I
THE CLASSIFICATION ACCURACY RATE OF THE THREAT MODELS AGAINST CLOSED-WORLD DATASETS. * THE MAXIMUM THEORETICAL ACCURACY AGAINST W-T IS 50%.

Threat Model	No Defense	WTF-PAD	W-T*
DNN	97.99%	80.65%	36.32%
CNN	99.93%	97.94%	49.52%

Table II
THE CLASSIFICATION ACCURACY RATE OF THE THREAT MODELS AGAINST OPEN-WORLD DATASETS. * THE MAXIMUM THEORETICAL ACCURACY AGAINST W-T IS 50%.

Threat Model	No Defense	WTF-PAD	W-T*
DNN	71.50%	31.33%	25.95%
CNN	96.93%	81.80%	37.52%

apply our scheme in the entry server (*i.e.,* bridge or proxy) which is a volunteer of the Tor network (See Figure 1).

C. Results

In order to provide a better understanding of our finding, this section is broken down into two main parts: 1) WF attacks evaluation and 2) DFD evaluation. We evaluated the performance of both deep learning-based WF attacks and DFD approach over both closed-world and open-world datasets.
Assumptions. We assume that the adversary can obtain all the network traffic from the client to Tor network and vice versa. Moreover, and as mentioned earlier, the adversary has only passive access to the network traffic.

1) WF Attacks Evaluation: In the following, we evaluate the performance of our deep learning-based WF attacks in the closed-world and open-world settings.
CWS. We built our DL-based WF attacks based on the collected network traces in the closed-world scenario. The closed-world traffic is categorized into *defended*, with WTF-PAD and W-T methods, and *undefended* traces. We trained CNN and DNN models for each of these categories. The performance of each of these WF attacks is listed in detail in Table I. As it can be seen, both CNN and DNN models are able to achieve high accuracy rates—99.93% and 97.99%, respectively—for the *undefended* dataset. Note that for the case of W-T any threat model can theoretically obtain at most 50% of accuracy rate [24]. Thus, the obtained result of 49.52% classification accuracy rate is quite reasonable. Although the performance of the DNN model drops for the *defended* datasets, the performance of CNN model remains high. This is quite justifiable given that the convoluted structure provides better feature extraction, thus classification accuracy.
OWS. We trained our CNN and DNN WF attacks in the open-world settings. All steps taken are similar to the closed-world setting, and the classification results are in Table II. In comparison to the closed-world scenario, the performance has decreased. This is because of the increased size of the data and accordingly the complexity of the classification task. Similar to the closed-world scenario, the performance of the CNN model is higher than DNN model in both the *defended* and the *undefended* datasets. This can be inferred as smaller confidence of the DNN model compared to the CNN model.

2) DFD Evaluation: Generally, machine/deep learning-based WF models learn the inherent pattern of the Tor users' online activity from his network traffic. Therefore, any potential defense mechanism should be able to break the pattern such that the adversary cannot identify the Tor users' destination. Such patterns can be broken by injecting dummy messages at the client-side only or both client and server sides.
OWI. In one-way injection, the dummy messages are injected in every outgoing burst. The injection is performed in either the client or the server. To observe which side injection has more impact on breaking the patterns, we performed two individual experiments by varying the perturbation rate (P).

- **Client-side Injection (CSI).** Here, we inject the dummy messages in each burst based on the length of the previous burst. We argue that injection of dummy messages using this approach is more logical. We evaluated the performance of this approach using closed-world and open-world *undefended* datasets. The misclassification rate of DFD using the different perturbation rates is shown in Figure 6 and Figure 7. As it can be seen, the higher the perturbation rate, the higher is the misclassification rate as well. For instance, DFD achieved a misclassification rate of 86.02% over the CNN model.
- **Server-side Injection (SSI).** Unlike client-side injection, here we injected dummy messages only on the server-side. The performance of this approach is evaluated based on both the closed-world and open-world settings. Figure 8 and Figure 9 demonstrate the misclassification rate of the DL-based WF attacks using different perturbation rates. We found that the misclassification rate increases at higher perturbation rates. Specifically, in the closed-world setting, the misclassification rate at 25% and 50% perturbation rate were 67.43% and 82.8%, respectively. The misclassification rate increases up to 94.52% using 100% perturbation rate over the CNN model. The same trend is seen in the open-world setting.

Observation. In these experiments, we found that the server-side injection significantly impacts the misclassification rate, in compare to the client-size injection.
TWI. The two-way injection approach is similar to the one-way injection, although dummy messages are injected at both the client and server. To observe the impact of each injection side (*i.e.,* client or server), we performed extensive experiments based on the symmetric and asymmetric injection.

- **Symmetric injection (SI)** Here, the percentage of incoming and outgoing dummy messages are equally changed from 1% to 25%. Figure 10 and Figure 11 show the misclassification of the two-way injection method for closed-world and open-world settings using six perturbation rates. We observed that the misclassification rate varies from 65.07% ($P = 1\%$) to 67.77% ($P = 25\%$) for the CNN model over the closed-world setting.

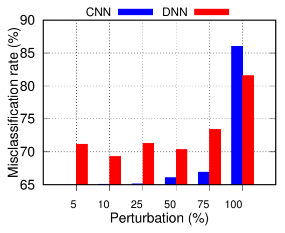


Figure 6. Misclassification rate of OWI-CSI in CWS.

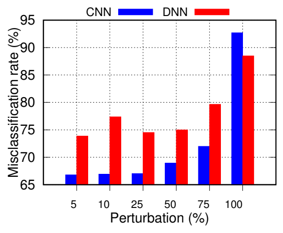


Figure 7. Misclassification rate of OWI-CSI in OWS.

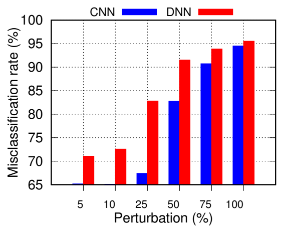


Figure 8. Misclassification rate of OWI-SSI in CWS.

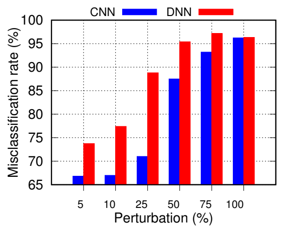


Figure 9. Misclassification rate of OWI-SSI in OWS.

- **Asymmetric injection (AI).** Unlike the symmetric injection, here, we fixed the percentage of either the client-side or the server-side's dummy message injection rate and explored the behavior of the model under different injection rates on the other side.
  Fixed towards client-side (FTCS). In this experiment, we fixed the perturbation rate at the client-side to 100%, while the injection rate at the server side is varied from 5% to 100%. Figure 14 and Figure 15 show the closed-world and open-world experiment results, respectively. We observed that there is a positive correlation between the misclassification and injection rates. We observed that as the injection rate at the client-side increases, the misclassification rate increases over the CNN model.
  Fixed towards server-side (FTSS). We fixed the perturbation rate at the server side to 15%, while the injection rate at the client side is varied from 5% to 100%. Figure 12 and Figure 13 depict the misclassification rate over the closed-world and open-world settings, respectively. We observed that as the injection rate at the client-side increases, the misclassification rate increases.

Observations. We observed that we could achieve almost the same misclassification rate of 67.77% using both 25% perturbation rate for one-way client-side and two-way symmetric injections. In addition, in the asymmetric experiment, we achieve a misclassification rate of 86.02% and 85.76% using 100% and 15% injection rates at the client and server-side, respectively. Where nearly the same misclassification rate can be achieved using one-way injection method at the client-side ($P = 100\%$). Therefore, we argue that one-way client-side injection is, in general, equivalent to two-way injection approach in terms of the misclassification rate.

Defense-aware Adversary. Similar to WTF-PAD and W-T, we evaluate our proposed approach performance in defending against adversaries fully aware of the used defense method, and the underneath machine learning architecture. To this end, we trained CNN and DNN models on closed-world samples generated from DFD (*i.e.,* two-way injection: client 15% and server 25%). The CNN and DNN models achieved a remarkable results of 97.37% and 76.91% in identifying traces websites, respectively. This performance is caused by the injection, as it depends on the original traces length. While the extracted patterns are not similar to the original patterns, deep learning models were able to learn and distinguish between websites. We observed similar results in the open-world setting, where the accuracy were 89.20% for CNN model and 73.05% for DNN model. To highlight this issue, we investigated the effect of changing the injection rate on the performance of the trained model. To do so, we applied DFD with two-way injection (client 25% and server 100%) on closed-world traces, achieving a detection accuracy of 12.25% and 3.36% for CNN and DNN trained models, respectively. Similarly, we achieved a detection accuracy of 17.47% for CNN model, and 53.37% for DNN models under same settings on open-world traces. Due to DFD design, and the ease of changing the injection rate (P), adapting a dynamic injection rate (*i.e.,* automatic update of P) prevents the exposure of the

Table III
CWS: BANDWIDTH OVERHEAD USING DIFFERENT CONFIGURATIONS. HERE, BO: BANDWIDTH OVERHEAD AND MR: MISCLASSIFICATION RATE.

Operation Method	P (Out)	P (In)	BO	MR
One-way (client-side)	50%	0%	7.21%	66.07%
	75%	0%	10.80%	66.93%
	100%	0%	14.43%	86.02%
One-way (server-side)	0%	50%	42.78%	82.80%
	0%	75%	64.17%	90.73%
	0%	100%	85.56%	94.52%
Two-way (symmetric)	5%	5%	5.00%	65.00%
	15%	15%	15.00%	65.79%
	25%	25%	25.00%	67.77%
Two-way (asymmetric)	50%	15%	20.05%	80.08%
	75%	15%	23.66%	81.15%
	100%	15%	27.27%	85.76%

Table IV
OWS:BANDWIDTH OVERHEAD USING DIFFERENT CONFIGURATIONS. HERE, BO: BANDWIDTH OVERHEAD AND MR: MISCLASSIFICATION RATE.

Operation Method	P (Out)	P (In)	BO	MR
One-way (client-side)	50%	0%	7.13%	68.95%
	75%	0%	10.69%	71.99%
	100%	0%	14.26%	92.71%
One-way (server-side)	0%	50%	42.86%	87.50%
	0%	75%	64.38%	93.20%
	0%	100%	85.73%	96.24%
Two-way (symmetric)	5%	5%	5.00%	66.98%
	15%	15%	15.00%	67.63%
	25%	25%	25.00%	71.72%
Two-way (asymmetric)	50%	15%	19.99%	89.19%
	75%	15%	23.55%	89.63%
	100%	15%	27.12%	92.66%

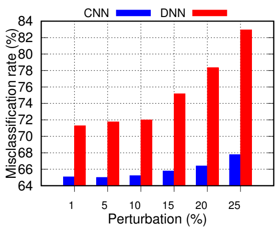


Figure 10. Misclassification rate of TWI-SI in CWS. CSI and SSI are changed equally.

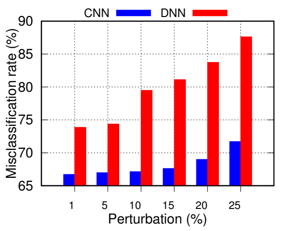


Figure 11. Misclassification rate of TWI-SI in OWS. CSI and SSI are changed equally.

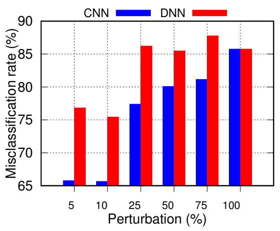


Figure 12. Misclassification rate of TWI-AI in CWS. SSI is fixed to 15%, while CSI is changed.

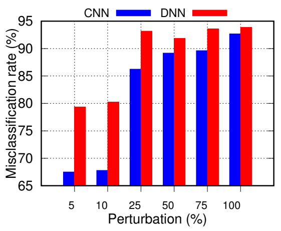


Figure 13. Misclassification rate of TWI-AI in OWS. SSI is fixed to 15%, while CSI is changed.

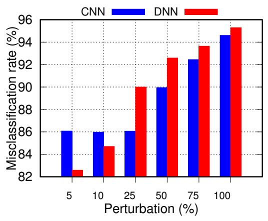


Figure 14. Misclassification rate of TWI-AI in CWS. CSI is fixed to 100%, while SSI is changed.

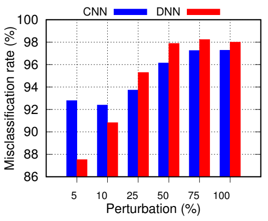


Figure 15. Misclassification rate of TWI-AI in OWS. CSI is fixed to 100%, while SSI is changed.

users' patterns, resulting in a secure website visiting.

D. Discussion

We introduced DFD to break the inherent pattern observed while visiting websites and used to break the privacy of Tor. In line with previous defenses, the message injection is carried out within bursts. However, DFD injects the messages over each burst, where the number of injected messages is decided by two factors: 1) the previous burst length and 2) perturbation rate. Injecting at each burst ensures the destruction of the existing patterns utilized by The deep learning model. Our evaluations show a high correlation between the misclassification rate and the perturbation rate.

Trade-off. Through our experiments, we conclude that the burst injection of both the client-side and server-side has a significant impact in terms of the misclassification rate. Although the increase of injection ratio leads to a higher misclassification rate, they cause a huge increase in bandwidth overhead, which degrades the user experience. In the one-way injection scenario, the outgoing bandwidth overhead is equivalent to the perturbation rate of outgoing traffic. Whereas, the overall bandwidth overhead of the two-way symmetric injection is the perturbation rate of the entire traffic (*i.e.,* incoming and outgoing). Obviously, two-way injection presents a higher bandwidth overhead compared to the one-way injection. To show the trade-off, we list the average bandwidth overhead and the corresponding misclassification rate for each injection mode (*i.e.,* one-way, two-way symmetric, two-way asymmetric) in Table III and Table IV. Moreover, as shown in Figure 16 and Figure 17, the fraction of the outgoing messages is way smaller than that of incoming messages for most traces, which enables us to apply higher injection rates at the client-side without significantly increasing the bandwidth overhead. This can be illustrated by the bandwidth overhead

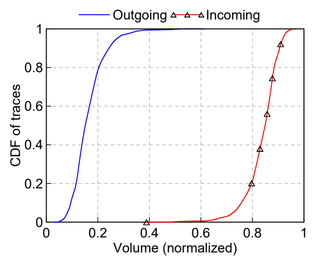


Figure 16. CDF of traces with outgoing/incoming volumes in CWS.

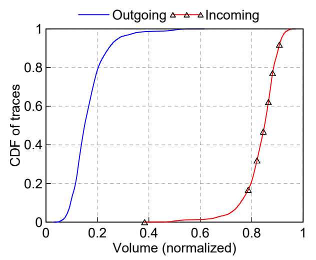


Figure 17. CDF of traces with outgoing/incoming volumes in OWS.

of 14.43% in 100% client-side perturbation rate, which is less than 15% two-way symmetric injection while providing 25.08% more of the misclassification rate. The same pattern can also be observed from the one-way server-side injection, where injecting at 50% rate will cause a bandwidth overhead of 42.86%. Overall, the one-way client-side injection with 100% perturbation rate presents the best misclassification to bandwidth overhead trade-off.

VI. Conclusion

In this paper, we introduced a novel Website Fingerprinting (WF) defense scheme, called Deep Fingerprinting Defender (DFD), to defend against deep learning-based attacks. In its heart, DFD carefully inject dummy messages within every traffic burst generated by the communication between the Tor client and server. DFD supports one-way and two-way injection modes for users to handle trade-off between misclassification rate and overhead on demand. In the open-world setting, DFD achieved 86.02% misclassification rate with only 14.43% of bandwidth overhead. Further, in the close-world setting, the misclassification rate was as high as 92.71% with the similar bandwidth overhead (14.26%). We evaluated the performance of DFD against adversaries with prior knowledge of our injection mechanism. We found that applying DFD with automatic update of the injection rate can mitigate the deep learning-based WF attacks effects, concealing the patterns and providing a secure website visiting behavior.

Acknowledgement

This research was supported by Global Research Laboratory Program by the National Research Foundation of Korea (NRF-2016K1A1A2912757) and a CyberFlorida Capacity Building Award.

References

[1] "Users-tor metrics." [Online]. Available: https://metrics.torproject.org/userstats-relay-country.html

[2] "Tor network status." [Online]. Available: http://torstatus.blutmagie.de/

[3] D. Herrmann, R. Wendolsky, and H. 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*, 2009, pp. 31–42.

[4] M. Juárez, S. Afroz, G. Acar, C. Díaz, and R. Greenstadt, "A critical evaluation of website fingerprinting attacks," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 263–274.

[5] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website fingerprinting at internet scale," in *Proceedings of the 23rd Annual Network and Distributed System Security Symposium, NDSS*, 2016.

[6] T. Wang and I. Goldberg, "On realistically attacking tor with website fingerprinting," *PoPETs*, vol. 2016, no. 4, pp. 21–36, 2016.

[7] P. Sirinam, M. Imani, M. Juárez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2018, pp. 1928–1943.

[8] V. Rimmer, D. Preuveneers, M. Juárez, T. van Goethem, and W. Joosen, "Automated website fingerprinting through deep learning," in *Proceedings of the 25th Annual Network and Distributed System Security Symposium, NDSS*, 2018.

[9] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for website fingerprinting." in *Proceedings of the USENIX Security Symposium*, 2014, pp. 143–157.

[10] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *Proceedings of the USENIX Security Symposium*, 2016, pp. 1187–1203.

[11] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for website fingerprinting," in *Proceedings of the 23rd USENIX Security Symposium*, 2014, pp. 143–157.

[12] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg, "A systematic approach to developing and evaluating website fingerprinting defenses," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 227–238.

[13] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright, "Toward an efficient website fingerprinting defense," in *Proceedings of the European Symposium on Research in Computer Security*, 2016, pp. 27–46.

[14] A. Addeh, A. Khormali, and N. A. Golilarz, "Control chart pattern recognition using rbf neural network with new training algorithm and practical features," *ISA transactions*, 2018.

[15] A. Mohaisen, O. Alrawi, and M. Mohaisen, "AMAL: high-fidelity, behavior-based automated malware analysis and classification," *Computers & Security*, vol. 52, pp. 251–266, 2015.

[16] J. Niu, Y. Liu, M. Guizani, and Z. Ouyang, "Deep cnn-based real-time traffic light detector for self-driving vehicles," *IEEE Transactions on Mobile Computing*, 2019.

[17] M. Abuhamad, T. AbuHmed, A. Mohaisen, and D. Nyang, "Large-scale and language-oblivious code authorship identification," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2018, pp. 101–114.

[18] M. Abuhamad, J. Rhim, T. AbuHmed, S. Ullah, S. Kang, and D. Nyang, "Code authorship identification using convolutional neural networks," *Future Generation Computer Systems*, vol. 95, pp. 104–115, 2019.

[19] H. Alasmary, A. Khormali, A. Anwar, J. Park, J. Choi, A. Abusnaina, A. Awad, D. Nyang, and A. Mohaisen, "Analyzing and detecting emerging internet of things malware: A graph-based approach," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8977–8988, 2019.

[20] A. Abusnaina, A. Khormali, H. Alasmary, J. Park, A. Anwar, and A. Mohaisen, "Adversarial learning attacks on graph-based iot malware detection systems," in *39th IEEE International Conference on Distributed Computing Systems, ICDCS*, vol. 10, 2019.

[21] A. Abusnaina, H. Alasmary, M. Abuhamad, S. Salem, D. Nyang, and A. Mohaisen, "Subgraph-based adversarial examples against graph-based iot malware detection systems," in *International Conference on Computational Data and Social Networks*, 2019, pp. 268–281.

[22] C. S. Ian J. Goodfellow, Jonathon Shlens, "Explaining and harnessing adversarial examples," in *Proceedings of the International Conference on Learning Representations.*, 2015.

[23] M. Imani, M. S. Rahman, and M. Wright, "Adversarial traces for website fingerprinting defense," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 2225–2227.

[24] T. Wang and I. Goldberg, "Walkie-talkie: An efficient defense against passive website fingerprinting attacks," in *Proceedings of the 26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1375–1390.

[25] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *Proceedings of the first ACM Cloud Computing Security Workshop, CCSW*, 2009, pp. 31–42.

[26] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society, WPES*, 2011, pp. 103–114.

[27] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: website fingerprinting attacks and defenses," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2012, pp. 605–616.

[28] T. Wang and I. Goldberg, "Improved website fingerprinting on tor," in *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES*, 2013, pp. 201–212.

[29] K. Abe and S. Goto, "Fingerprinting attack on tor anonymity using deep learning," *Proceedings of the Asia-Pacific Advanced Network*, vol. 42, pp. 15–20, 2016.

[30] D. Lu, S. Bhat, A. Kwon, and S. Devadas, "Dynaflow: An efficient website fingerprinting defense based on dynamically-adjusting flows," in *Proceedings of the 2018 Workshop on Privacy in the Electronic Society*, 2018, pp. 109–113.

[31] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, I still see you: Why efficient traffic analysis countermeasures fail," in *Proceedings of the IEEE Symposium on Security and Privacy, S&P*, 2012, pp. 332–346.

[32] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg, "A systematic approach to developing and evaluating website fingerprinting defenses," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 227–238.

[33] X. Cai, R. Nithyanand, and R. Johnson, "Cs-buflo: A congestion sensitive website fingerprinting defense," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, 2014, pp. 121–130.

[34] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *IEEE European Symposium on Security and Privacy, EuroS&P*, 2016, pp. 372–387.

[35] A. Abusnaina, A. Khormali, D. Nyang, M. Yuksel, and A. Mohaisen, "Examining the robustness of learning-based ddos detection in software defined networks," in *2019 IEEE Conference on Dependable and Secure Computing (DSC)*. IEEE, 2019, pp. 1–8.

[36] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, 2011, pp. 103–114.

[37] "A critique of website traffic fingerprinting attacks," Nov 2013. [Online]. Available: https://blog.torproject.org/critique-website-traffic-fingerprinting-attacks

[38] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[39] "Deep learning and machine learning differences: Recent views in an ongoing debate," Mar 2017. [Online]. Available: http://www.dataversity.net/deep-learning-machine-learning-differences-recent-views-ongoing-debate/

[40] V. Shmatikov and M. Wang, "Timing analysis in low-latency mix networks: Attacks and defenses," in *Proceedings of the 2006 European Symposium on Research in Computer Security, ESORICS*, 2006, pp. 18–33.

[41] M. Allman, "On the generation and use of tcp acknowledgments," *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 5, pp. 4–21, 1998.

[42] K. Liu and J. Y. Lee, "On improving tcp performance over mobile data networks," *IEEE Transactions on Mobile Computing*, vol. 15, no. 10, pp. 2522–2536, 2016.