Adv-DWF: Defending Against Deep-Learning-Based Website Fingerprinting Attacks with Adversarial Traces

Mohsen Imani
Qualys
iman.moh@gmail.com

Mohammad Saidur Rahman*
Center for Cybersecurity
Rochester Institute of Technology
saidur.rahman@mail.rit.edu

Nate Mathews
Center for Cybersecurity
Rochester Institute of Technology
nate.mathew@rit.edu

Aneesh Yogesh Joshi
Center for Cybersecurity
Rochester Institute of Technology
aj4524@rit.edu

Matthew Wright
Center for Cybersecurity
Rochester Institute of Technology
matthew.wright@rit.edu

Abstract
Website Fingerprinting (WF) is a type of traffic analysis attack that enables a local passive eavesdropper to infer the victim’s activity even when the traffic is protected by encryption, a VPN, or some other anonymity system like Tor. Leveraging a deep-learning classifier, a WF attacker can gain up to 98% accuracy against Tor. Existing WF defenses are either too expensive in terms of bandwidth and latency overheads (e.g. 2-3 times as large or slow) or ineffective against the latest attacks. In this paper, we explore a novel defense, Adv-DWF, based on the idea of adversarial examples that have been shown to undermine machine learning classifiers in other domains. Our Adv-DWF defense adds padding to a traffic trace in a manner that fools the classifier into classifying it as coming from a different site. The technique drops the accuracy of the state-of-the-art attack augmented with adversarial training from 98% to 35%, while incurring a reasonable 56% bandwidth overhead. For most of the cases, the state-of-the-art attack’s accuracies of our defense is at least 45% and 14% lower than state-of-the-art defenses WTF-PAD and Walkie-Talkie (W-T), respectively. The Top-2 accuracy of our defense is at best 56.9%, while it is over 98% for W-T. In addition, for the most cases, the bandwidth overheads of our defense is at least 8% and 6% lower than WTF-PAD and W-T, respectively, showing its promise as a possible defense for Tor.

1 Introduction
The Tor anonymity system is known to be vulnerable to traffic analysis attacks. One such attack is Website Fingerprinting (WF), which enables an eavesdropper between the client and the first Tor node on her path to identify which websites the client is visiting. Figure 1 shows a WF attack model in the Tor network. This local passive adversary could be sniffing the client’s wireless connection, have compromised her cable/DSL modem, or gotten access to the client’s ISP or workplace network.

The WF attack is effectively a supervised classification problem, in which the website domain names are labels and each traffic trace is an instance to be classified or to be used for training. The state-of-the-art WF attack, deep fingerprinting (DF), is based on a convolutional neural network (CNN) which is one of the powerful deep learning models to identify the hidden pattern from the dataset. The DF attack can successfully reach up to 98% accuracy to distinguish a site in an undefended dataset in a closed-world setting [29], and it has high precision and recall in the more realistic open-world setting.

In response to the threat of WF attacks, there have been numerous defenses proposed [34, 17, 25, 18, 3, 4, 14, 33]. WF defenses try to change the pattern of the traffic in a way that confounds the classifier. Tor traffic is already divided into fixed-sized cells of 512 bytes, and the order of objects requested from a site is randomized. The remaining ways to modify traffic are to add padding packets and to delay some packets. The BuFLO family of defenses (including BuFLO [5], CS-BuFLO [3], and Tamaraw [4]) apply these techniques and are effective, but require high overheads and make loading a website take two or three times as long as in Tor. Recently, two relatively lightweight defenses, WTF-PAD and Walkie-Talkie (W-T), have been proposed. WTF-PAD [14] offers lower overheads and is considered to be a leading candidate for deployment in Tor [26]. However, Sirinam et al. show that WTF-PAD is badly undermined by the DF attack, which achieves over 90% accuracy in the closed-world. The recently-proposed Walkie-Talkie [33] (W-T) performs relatively better against DF attack. However, the Top-2 accuracy of W-T defense against DF attack is above 98% which provides an attacker the ability to scope down the potential two sites [29].

In this paper, we introduce a new defense strategy using adversarial examples generated by a deep neural network. Adversarial examples show tremendous success in fooling deep-learning based models such as CNN to misclassify with...
100% accuracy [6]. As DF attack is based on a CNN model, we want to fool this classifier to misclassify the websites with adversarial traces. The success rate of adversarial examples in computer vision is measured by testing the classification accuracy of adversarial examples in a deep-learning model which is trained with original samples. However, WF defense provides the attacker access to the defense mechanism and enables him to train his classifier with adversarial examples. This might lead him to achieve high testing accuracy to classify the websites even though those are manipulated with adversarial examples. We, thus, propose a novel approach to modify website traces that is suited and can be effective for WF defense which can cause misclassification in a deep-learning based classifier with moderate amounts of bandwidth overhead. From our experiments, we find that our defense can significantly decrease the accuracy of state-of-the-art attack from 98% to 35% with a 56% bandwidth overhead for full-duplex network traffic without requiring additional delays.

The key contributions of this work are as follows:

- We are the first to leverage the concept of adversarial examples in defending anonymous network traffic from WF attacks.
- We modify the state-of-the-art algorithms in generating adversarial examples in the computer vision to our needs and show that they fail as a defense in a WF attack scenario.
- We describe a novel algorithm, Adv-DWF, to generate adversarial traces that can fool a deep-learning classifier. The algorithm shows how to modify a trace with padding packets to cause the classifier to misidentify the site.
- We evaluate our method and find that it significantly reduces accuracy of the state-of-the-art WF attack from 98% to 35%, with 56% bandwidth overhead and no added delays.
- We show that it is harder for an attacker to scope down to top two sites in our defense. The state-of-the-art attack can reach up to 56.9% Top-2 accuracy against our defense while the Top-2 accuracy of W-T is over 98%.

2 Threat & Defense Model

2.1 Threat Model

We assume that the client browses the Internet through the Tor network to hide her activities (see Figure 1). The adversary of interest is local, which means the attacker is positioned somewhere in the network between the client and Tor guard node. The attacker is assumed to already know the identity of the client. His goal is to detect the websites that the client is visiting. A local adversary can be an eavesdropper on the user’s local network, local system administrators, Internet Service Providers, any networks between the user and the entry node, or the operator of the entry node. The attacker is passive, meaning that he only observes and records the traffic traces that pass through the network. He does not have the ability to drop, delay, or modify real packets in the traffic stream.

In a website fingerprinting (WF) attack, the attacker feeds the collected network traffic into a trained machine-learning or deep-learning classifier. For the purpose of training, the WF attacker needs to collect traffic of various sites as a client of Tor network. The attacker must deal with the limitation that it is not feasible to collect the network traffic of all the sites on the web. To address this, the attacker identifies a set of monitored sites that he wants to track. The attacker limits the scope of his attack to the identification of any website visits that are within the monitored set. The set of all other sites which is known as the unmonitored set.

WF attacks and defenses are evaluated in two different settings: closed-world and open-world. In the closed-world setting, we assume that the client is limited to visiting only the monitored sites. The training and testing set used by the attacker only include samples from the monitored set. The closed-world scenario models an ideal setting for the attacker and is not indicative of the attack’s real-world performance. From the perspective of developing a WF defense, defeating a closed-world attack means defeating the attacker from the most advantageous situation.

In contrast, the open-world scenario models a more realistic setting in which the client may visit websites from both the monitored and unmonitored sites. In this setting, the attacker trains on the monitored sites and a representative (but not comprehensive) sample of unmonitored sites. The open-world classifier is then evaluated against both monitored and unmonitored sites, where the set of unmonitored sites used for testing does not intersect with the training set.

2.2 Defense Model

The purpose of a WF defense is to make the traffic trace indistinguishable from traces from other sites. To achieve this, the real traffic stream must be manipulated in some way. Be-
cause traffic is bidirectional, the deployment of a successful WF defense requires participation from both the client and a cooperating node. It is difficult, if not impossible, for the client to defend their traffic alone as they have control over only outgoing traffic in the communication stream. To defend both directions in the network stream, the client must have the cooperation of one of the nodes in its Tor circuit. We call this cooperating node the bridge. The bridge must be located somewhere between the adversary and the client’s destination server so that the adversary only has access to the obfuscated traffic stream. In order to reduce additional bandwidth requirements on the full Tor circuit, it is usually ideal that the bridge be located on the guard node.

3 Background & Related Work

3.1 WF Attacks

The study of website fingerprinting dates back to the 1990s when the security researchers first investigated information leak in encrypted HTTP requests which could expose the URL of the site being visited [31]. The threat of website fingerprinting against Tor was first evaluated by Herrmann et al. [12] in 2009. Their attack used a Naive Bayes classifier with a feature set created from the distribution of the frequencies of packet lengths. However this reliance on packet length lead to the attacks failure, reaching only 3% accuracy. Tor sends data in fixed 512-byte units, known as cells, which makes packet length features ineffective. Since then, researchers have incrementally improved the performance of WF attacks through the development of better feature sets and the application of new classification algorithms. In this section we explore several notable attacks which utilize hand-crafted features as well as recent deep-learning based attacks.

**k-NN:** The k-NN attack introduced by Wang et al. [32], uses a modified k-nearest neighbors (k-NN) classifier. Instead of using the Euclidean distance, the attack uses a weighted distance function to compare samples. The k-NN attack works in two phases: the weight-learning phase and the classification phase. The weight-learning process is an iterative algorithm in which weights are learned from traffic instances in the training dataset. Weights indicate the importance of each feature in computing the distance. In the classification phase, the learned weights are used to compute the distances and identify the nearest neighbors. One of the most important contributions of this work is the feature set. The authors defined a diverse set of features that includes traffic features such as packet ordering, concentration of the packets, and bursts in the traffic. Using this large feature set with weight-adjusting k-NN, they obtained 91% accuracy in a 100-site closed-world setting.

**CUMUL:** Panchenko et al. [21] proposed the CUMUL attack, which uses Support Vector Machines (SVM) as the classifier. Their novel feature set consists of the sequence of the cumulative sums of packet sizes. They also collected a dataset that is more representative of real sites browsed on the Internet. Panchenko et al. achieved this by assembling their list of websites from different sources—trend links in Twitter, trends in Google, and censored sites in China—which may more appropriately model the types of traffic passing through the Tor network. Moreover, they were the first to differentiate website fingerprinting from webpage fingerprinting. They evaluated the fingerprintability of both single webpages and complete websites. Their webpage fingerprinting attack obtained 92% accuracy in the closed-world setting. Sirinam et al. [29] shows that CUMUL attack has comparable performance with DF attack. Thus, to evaluate the effectiveness of our WF defenses, we selected the CUMUL attack as one of the WF attacks to test our defense against.

**k-Fingerprinting (k-FP):** The k-FP attack was proposed by Hayes and Danezis [11]. These researchers developed a comprehensive feature set by aggregating all the previously defined features together. Their feature set also included some new features, such as statistics on the timestamps and the volume of the traffic. The attack used Random Forests to rank and identify the most important features in the traffic. The classifier first learns the most important features as the fingerprints of a site using Random Forests. Later the fingerprints are used by a k-NN classifier to perform the classification task. Their attack reaches 91% accuracy in the closed-world setting.

**SDAE:** Recently, deep learning techniques have attracted the attention of privacy researchers due to their excellent performance in image recognition tasks. The first to investigate their usage for WF was Abe and Goto [2] who developed an attack based on Stacked Denoising Autoencoders (SDAE). Unlike prior work, their attack did not utilize handcrafted features. Rather, their model was trained on the raw packet representation, defined by a sequence of positive and negative values. Despite this innovation, their attack achieved a lower accuracy rate than the previous state-of-the-art at only 88% in the closed-world setting. The reason of their low accuracy rate would seem to be due to the limited number of samples in the dataset (100 per site) as DL models require more data to train when compared to other classification algorithms.

**Automated Website Fingerprinting (AWF):** Rimmer et al. [22] proposed using deep learning to bypass the feature engineering phase of traditional WF attacks. To more effectively utilize DL techniques, they collected a larger dataset of 900 sites with 2,500 trace instances per site. They applied several different DL architectures—SDAE, Convolutional Neural Network (CNN), and Long Short-Term Memory (LSTM)—on the traffic traces. They found that their CNN model outperforms the other DL models they developed.
Deep Fingerprinting (DF): Sirinam et al. [29] further developed a deep CNN model that could outperform all the previous models, reaching up to 98% accuracy rate in the closed-world setting. They evaluated their attack against a dataset of 100 sites with 1,000 instances each. They also examined the effectiveness of their model against WF defenses, where they showed that their model can achieve concerning high performance against even with defended traffic. Most notably, their attack achieved 90% accuracy against WTF-PAD [14], a potential WF defense candidate to be deployed on Tor.

3.2 WF Defenses

In an effort to defeat WF attackers, researchers have crafted various defenses that generate cover traffic to hide the features present in the website traffic. WF defenses are able to manipulate the traffic stream with two operations: delaying real packets and sending dummy packets. These manipulations, however, come at a cost: sending dummy packets adds an additional bandwidth overhead to the network, while delaying packets causes a latency overhead which directly impacts the time required to load the page. Several studies have thus tried to balance the trade-off between the WF defense’s overhead and efficacy of the defense against WF attacks. In this section, we review these WF defenses.

Constant-rate padding defenses: This family of defenses transmits traffic at a constant rate in order to normalize trace characteristics. BuFLO [9] is the first defense of this kind and it sends the packets in the same constant rate in both directions. The defense ends transmission after the page has finished loading and a minimum amount of time has passed. The overhead of the traffic is governed by both the transmission rate and the minimum time threshold for the stopping condition. Moreover, although the defense covers fine-grained features like burst information, the coarse-grained features like the volume and load time of the page still leak information about the website. Tamaraw [4] and CS-BuFLO [3] extend the BuFLO design with the goal of addressing these issues. Since nearly all sites need more traffic to the client than they require, Tamaraw and CS-BuFLO transmit the download and upload packets at different fixed rates. To provide better cover traffic, after the page is loaded, Tamaraw keeps padding until the total number of transmitted bytes is a multiple of a fixed parameter. Similarly, CS-BuFLO pads the traffic to a power of two, or to a multiple of the power of the amount of transmitted bytes. BuFLO family defenses are expensive in terms of overhead, and they add 2 or 3 times latency overhead and more than 100% bandwidth overhead.

Supersequence defenses: This family of defenses depends on finding a supersequence for traffic traces. To do this, these defenses first cluster websites into anonymity sets and then find a representative sequence for each cluster such that it contains all the traffic sequences. All the websites that belong to the same cluster are then molded to the representative supersequence. This family includes Supersequence [32], Glove [19], and Walkie-Talkie [33]. Supersequence and Glove use approximation algorithms to estimate the supersequence of a set of sites. The traces are then padded in such a way so as to be equivalent to its supersequence. However, applying the molding directly to the cell sequences creates high bandwidth and latency costs. Walkie-Talkie (WT) differs from the other two defenses in that it uses anonymity sets of just two sites and traces are represented as burst sequences rather than cell sequences. Even with anonymity sets of size two, this produces a theoretical maximum accuracy of 50% .

Adaptive Padding (AP): Shmatikov and Wang [28] proposed Adaptive Padding (AP) as a countermeasure against end-to-end traffic analysis. Juarez et al. [14] extended the idea of AP and proposed the WTF-PAD defense as an adaptation of AP to protect Tor traffic against WF attacks. WTF-PAD tries to fill in large delays between packets (inter-packet arrival times). Whenever there is a large inter-packet arrival time, WTF-PAD sends a fake burst of dummy packets. This approach does not add any artificial delays to the traffic. Juarez et al. show that WTF-PAD can drop the accuracy of the k-NN attack from 92% to 17% with a cost of 60% bandwidth overhead. Wang & Goldberg report 31% bandwidth overhead and 34% latency overhead due to the use of half-duplex communication. Then Sirinam et al. [29] achieved 49.7% attack accuracy against WT.

Application-level defenses: Cherubin et al. [8] propose the first WF defenses designed to work at the application layer. They proposed two defenses in their work. The first of these defenses, ALPaCA, operates on the webserver of destination websites. This defense was specifically designed for special websites known as onion sites (further explained in Section 3.3). Cherubin et al. argued their defense was particularly suited for these sites because onion sites are typically more interested in preserving anonymity and would thus be more willing to run a defense. ALPaCA works by altering the size distribution for each content type, e.g. PNG, HTML, CSS,
to match the profile for an average onion site. In the best case, this defense has 41% latency overhead and 44% bandwidth overhead, and reduces the CUMUL attack’s accuracy from 56% to 33%. Their second defense, LLaMA, instead operates on the client exclusively. This defense adds random delays to HTTP requests in an effort to affect the order of the packets by manipulating HTTP request and responses patterns. LLaMA drops the accuracy of the CUMUL attack on Onion Sites from 56% to 34% at cost of 9% latency overhead and 7% bandwidth overhead.

3.3 Onion Sites

An onion site, formerly known as a hidden service, is an anonymized service provided through Tor. The locations and the IP addresses of onion servers are not known by Tor nodes nor their clients. Onion sites provide various kinds of services such as web publishing, messaging, and chat. These sites are particular interesting in that the services they provide are often very sensitive in nature.

Recent studies [20, 15] have shown that onion sites’ traffic can be easily distinguished from other types of Tor traffic. Therefore, the adversary can easily filter out the onion sites’ traffic from the rest of Tor traffic and specifically target these sites in a WF attack. Given that the number of onion sites is very small compared to that of the regular Web, the WF attacker is dealing with a small open world. Thus, the threat of WF against onion sites is particularly serious. As such this is a strong motivational force for the timely development of reliably effective WF defenses.

4 Defense Design

4.1 Adversarial Examples

Adversarial examples have become a matter of concern since Szegedy et al. [30] first demonstrated the vulnerability in several ML and DL models. These models misclassify the adversarial examples with a high confidence. For example, Papernot et al. [22] show that adversarial images can cause a targeted deep neural network to misclassify with 84.24% rate. Adversarial examples are inputs that are crafted from the distribution of correctly classified samples, but with slight perturbations that result in a misclassification. In many cases, adversarial examples can be general and transferable in that the adversarial examples crafted for a certain model may also be misclassified in other models that were trained on different subsets of the data.

The goal of creating adversarial examples is to modify samples from one class to make them be misclassified to another class where the amount of modification is limited. More precisely, given an input sample \(x\) and target class \(t\) such that the class of \(x\) and the target class are not the same \(C^*(x) \neq t\). The goal is to find \(x'\) which is close to \(x\) according to some distance metrics and \(C(x') = t\). In this case, \(x'\) is known as a targeted adversarial example since it is misclassified to a particular target label \(t\). The goal in WF defense is to generate an untargeted example \(x\) that is misclassified to any other classes except the true class (\(C^*(x)\)).

The reason behind the existence of adversarial examples is a mystery. There is some speculation about the cause of adversarial examples, such as the extreme non-linearity of deep neural networks, overfitting, insufficient model averaging, and insufficient regularization. Due to the effectiveness of adversarial examples, there is an arms race between defenses against adversarial examples and new types of adversarial example attacks. Carlini et al [6] showed that almost none of the recent defenses against adversarial examples are effective.

Based on the fact that adversarial examples are transferable and are resistant to mitigation attempts, they may be a good basis for a defense against WF attacks. The goal of WF defenses is to insert dummy packets to the traffic to fool the WF attacker’s classifier. The number of dummy packets in the traffic should be as small as possible. The goals (low-overhead and high-rate of misclassification) of WF defenses correspond to the properties of the adversarial examples that are generated with small perturbations to fool the classifier.

4.2 Challenges

In contrast to their typical applications, adversarial examples in WF must be generated from Web traffic traces. This is different than adversarial examples in other fields such as image classification, voice recognition, and text classification, and thus makes the use of existing techniques in generating adversarial examples ineffective for WF defense. The main challenges we face in crafting adversarial examples for web traffic traces are as follows:

- **Crafting Adversarial Examples on the Fly**: To craft the adversarial examples from traffic traces, we have to deal with the transmission of packets. The traffic trace is captured packet by packet and the adversarial examples should be generated in real time so that we can send and receive the packets simultaneously. As the entire trace is not accessible to the algorithm during the application of the defense, the adversarial examples should be generated on the fly.

- **No Packet Dropping, only Insertion**: The perturbation to the traffic traces can be done only through the sending of the dummy packets. Dropping real packets to perturb the traffic trace is not recommended due to network retransmissions that lead to higher bandwidth and latency overhead, and harming the user’s experience.

- **Two Parties are Involved**: Traffic traces are a combination of download and upload streams. Neither the client
nor the server has a full control of both streams. Therefore, the WF defense has two elements to control both sides of the traffic: the client side and a server in the network. Generating the adversarial traces in the burst level enables us to generate them in the server and send them to the client.

### 4.3 Using Existing Methods

![Figure 2: A visualization of bursts, with six outgoing bursts interspersed with six incoming bursts.](image)

In order to tackle the challenges mentioned in Section 4.2, we first model the traffic stream as a sequence of incoming (server to client) and outgoing (client to server) bursts. A burst is a sequence of consecutive packets in the same direction (see Figure 2). We can use the traffic traces of full-duplex communication and identify the burst sequence. Alternatively, we can use half-duplex communication as proposed for use in Walkie-Talkie to convert the traffic from a sequence of incoming and outgoing packets to a sequence of bursts. By using the Walkie-Talkie definition of bursts, we can increase the length of the burst by sending the dummy packets in the direction of the burst. Moreover, we can decrease the length of the burst by sending dummy packets in the opposite direction of an ongoing burst. We cannot, however, decrease the size of the real bursts.

Several different algorithms have been proposed for generating adversarial examples within the field of computer vision, including the Fast Gradient Sign Method (FGSM) [10], the Jacobian-Based Saliency Map Attack (JSMA) [23], and optimization-based methods [6] [16]. For our initial exploration of adversarial examples for WF defense, we examined the technique proposed by Carlini and Wagner.

In 2017, Carlini and Wagner proposed [6] a powerful optimization-based algorithm to generate adversarial examples. This attack was shown to defeat the state-of-the-art defense, known as defensive distillation [24]. Their attack algorithm is successful with 100% probability in both defended and undefended models. We modified their technique to suit our needs to generate adversarial traces out of the burst sequences.

Given a sample $x$ and a model $F$, the algorithm finds a perturbation $\delta$ that makes $x' = x + \delta$ to be misclassified to any other class than $C(x)$ ($C(x) = \text{argmax}_y F(x)_y$), or in the case of targeted attack it is classified to target class $t$. The algorithm tries to find $x'$ that is similar to $x$ based on distance metric $D$. The distance metrics can be an $L_p$-norm such that $L_0$, $L_2$, or $L_{\infty}$. The algorithm is formulated as:

$$\min \ | \ | \delta \ | \ |_p + c \cdot f(x + \delta)$$

such that $x + \delta \in [0, 1]^n$ (1)

The algorithm will find $\delta$ such that it minimizes the distance metric, which here is $L_p$ norm, and the objective function $f(x + \delta)$. $c$ is a constant used to scale both the distance metric and objective function in the same range. Carlini and Wagner [6] used binary search to find the proper value for $c$. They explored several objective functions and found that the following objective functions generate the best-performing loss functions:

For a targeted attack scenarios with target class $t$:

$$f(x') = \max_{i \neq t} (F(x')_i) - F(x')_t$$

(2)

For non-targeted attack scenarios where the true class for sample $x$ is class $y$, the objective function is:

$$f(x') = F(x')_y - \max_{i \neq y} (F(x')_i)$$

(3)

We evaluated the performance of this technique in two different WF attack scenarios: Defense-after-attack and Attack-after-defense. The defense-after-attack scenario represents the scenario most typically seen in the adversarial example literature in which the classifier has not been trained on any adversarial instances. In this scenario, we found that the accuracy of DF, the state-of-the-art deep-learning attack, was reduced from 98% to 3%. This scenario is however not realistic as it is likely (and usually assumed) that the attacker can discern what type of defense is in effect and train on representative samples. This scenario was examined in the attack-after-defense evaluations. When evaluated under this scenario, the Carlini et al. technique shows completely ineffectiveness with 97% attack accuracy. The full details of these evaluations can be found in Appendix A.

The results of this evaluation led to a very important insight: the scenario in which the effectiveness of adversarial examples are typically evaluated is notably different than that of a WF defense. Thus, the techniques that excel at producing adversarial examples for traditional attacks are poorly suited for our problem. In response to this discovery, we focused our efforts on the development of a new technique designed specifically for our needs. We discuss our method in the following section.

### 4.4 Generating Adversarial Traces

We now introduce, $\text{Adv-DWF}$, a novel algorithm to generate adversarial traces which more reliably fool the classifier. The ultimate goal is to cause the classifier to label a traffic
trace as coming from any other site. To defend a given trace (the source sample), Adv-DWF randomly picks a target sample from a pool of target samples. This pool can have samples, i.e. traffic traces, from either a closed-world dataset or a open-world dataset. We call this randomly selected sample from the pool the target sample. The closed-world target pool contains samples drawn randomly from the closed-world dataset. Any samples from the source sample’s class are excluded. Similarly, the open-world target pool contains samples drawn randomly from the open world dataset.

Adv-DWF gradually changes the source sample in a direction to get closer to the target sample. The amount of change applied to the source sample governs the bandwidth overhead of Adv-DWF. In order to limit the amount of change added to the traces, Adv-DWF changes the source samples until they are no longer classified as their source classes. Adv-DWF uses a trained classifier, called a detector, to evaluate whether the perturbed source sample will be misclassified. If the detector identifies the perturbed source sample is still in the source class, Adv-DWF keeps changing the source sample toward the target sample until it leaves the source class. In the following section we explain the Adv-DWF algorithm in detail.

4.5 Adv-DWF

We assume that we have a set of sensitive sites \( S \) that we want to protect. We train a detector, \( f(x) \), on a set of data from \( S \). We discuss the cases whether \( f(x) \) should be trained on only sensitive sites or both sensitive and non-sensitive sites in Section 5. We consider traffic trace \( I_t \) as an instance of source class \( s \in S \). Our goal is to alter \( I_t \) such that it is classified to the target class \( t \), \( t = f(I_t) \) and \( t \neq s \).

\( I_t \) is a sequence of the bursts, \( I_t = [b^0_t, b^1_t, ..., b^n_t] \). The only allowed operation on a burst, \( b^i_t \), is to add some positive values, \( \delta^i_t > 0 \), to that burst, \( b^i_t' = b^i_t + \delta^i_t \). The reason for using \( \delta^i_t > 0 \) is that we want to only increase the volume of the bursts. If \( \delta^i_t < 0 \), that would mean we should drop some packets to reduce the burst volume, but dropping real packets means losing data and requires re-transmission of the dropped packet.

To protect source sample \( I_s \), we pick \( p \) random samples from other classes, \( P_I = [P^0_{I_s}, P^1_{I_s}, ..., P^n_{I_s}] \). \( P^j_I \) is the target pool for \( I_s \). \( P^j_I \) is the \( j \)-th sample in the target pool and belongs to target class \( t \neq s \). We want to pick a target sample and re-cast the source sample to be classified as that target class. To decrease amount of change to the source sample, since adding padding adds bandwidth overhead, we pick the sample from the target pool that is closest to the source sample. We use the \( l^2 \) norm distance to find the closest target sample to the source sample (see Equation 1). Then we modify the source sample to decrease the distance to this target sample.

\[
I_t = \text{argmin}_{I_t \in P_I} D(I_s, I_t) \tag{4}
\]

\[
D(x, y) = l^2(x - y) \tag{5}
\]

To make the source sample leave the source class, we change it in the direction of the nearest sample \( (I_t) \) using a minimum amount of perturbation. We define \( \Delta \) as the perturbation vector that we will add to the source sample to generate its defended form \( I_{s_{\text{new}}} \).

\[
\Delta = [\delta_0, \delta_1, ..., \delta_n] \quad (\forall i \in [0, ..., n]: \delta_i > 0) \tag{6}
\]

\[
I_{s_{\text{new}}} = I_s + \Delta \tag{7}
\]

In order too find a \( \Delta \) that minimizes the overhead, we aim to minimize distance \( D(P^s_{\text{new}}, I_t) \). To do so, we compute the gradient of the distance with respect to the input. The gradient points in the direction of steepest ascent, which would maximize the distance. Therefore, we compute the gradient of the negative of the distance with respect to the input, and we move the source sample in that direction towards the target sample. In particular:

\[
\nabla(-D(I_t, I_{t_{\text{new}}})) = -\frac{\partial D(I, I_{t_{\text{new}}})}{\partial I} = \left[ -\frac{\partial D(I, I_{t_{\text{new}}})}{\partial b_{i_t}} \right]_{i \in [0, ..., n]} \tag{8}
\]

Where \( b_i \) is the \( i \)-th burst in input \( I \).

To modify the source sample, we change bursts such that their corresponding values in \( (-D(I_t, I_{t_{\text{new}}})) \) are positive. Our perturbation vector \( \Delta \) is:

\[
\delta_t = \begin{cases} 
-\alpha \times \frac{\partial D(I_{t_{\text{new}}})}{\partial b_{i_t}} & \text{if } \frac{\partial D(I_{t_{\text{new}}})}{\partial b_{i_t}} > 0 \\
0 & \text{if } \frac{\partial D(I_{t_{\text{new}}})}{\partial b_{i_t}} \leq 0 
\end{cases} \tag{9}
\]

where \( \alpha \) is the parameter that amplifies the output of the gradient. The choice of \( \alpha \) has an impact on the convergence and the bandwidth overhead. If we pick a large value for \( \alpha \), we will take bigger steps toward the target sample and we will add more overhead. We modify the source sample by summing it with \( \Delta \), \( I_{s_{\text{new}}} = I_s + \Delta \). We iterate this process, computing \( \Delta \) for each \( I_s \) and updating the source sample until we leave the source class, \( f(I_{s_{\text{new}}}) \neq s \) or the number of iterations passes the maximum allowed iterations. Leaving the source class means that we have less confidence on the source class. So we fix a threshold value (\( \tau_c = 0.01 \)) for measuring the confidence. If the confidence of the detector on the source class is less than the threshold, Adv-DWF will stop changing the source sample (\( I_s \)).

As we only increase the bursts where \( \frac{\partial D(I_{t_{\text{new}}})}{\partial b_{i_t}} > 0 \), we may run into cases that after some iterations \( \nabla(-D(I_t, I_{t_{\text{new}}})) \) does not have any positive values or all the positive values are
extremely small such that they do not make any significant changes to \( I_s \). In such cases, if \( I_s^{\text{adv}} - I_s \) is smaller than a threshold \( T_D = 0.0001 \) for a few consecutive iterations (we used 10 consecutive iterations), and we are still in the source class, we drop the previous target samples in the target pool and fill it with new target samples and continue the process.

5 Evaluation

5.1 Datasets

We apply our algorithm for generating adversarial examples on the traffic traces on the burst level. We define the bursts as the sequence of the consecutive packets in the same direction. We can get the burst sequence of the traffic traces from both full-duplex and half-duplex communication modes. Walkie-Talkie (WT) [33] works on the half-duplex communication and it finds the supersequence in the burst level. In our experiments, we use burst sequences of both full-duplex and half-duplex datasets.

**Full-Duplex** Sirinam et al. [29] collected a reasonably large dataset of the full-duplex traffic traces. Their dataset has 95 classes with 1000 instances each. 95 sites are from the top 100 sites in Alexa.com [1]. We further process their data by removing any instances with less than 50 packets and the instances that start with incoming packets. After this processing, we end up with 518 instances for each site. We use this dataset for full-duplex closed-world experiments.

**Half-Duplex** Sirinam et al. [29] collected a big dataset of traffic traces over the half-duplex mode as well. Their dataset contains 100 sites which are also from top 100 sites in Alexa.com [1], with 900 instances for each class. For our half-duplex evaluation, we use their dataset. We cleaned their data and removed the traces shorter than 50 packets as well as traces which began with incoming packets. After the cleaning process we ended up with 83 classes with 720 instances per class for a total of 59760 traces. Moreover, they also have open world data of 40,000 sites with 1 instance each. We use both half-duplex closed world and open world datasets for our half-duplex closed-world evaluations.

In order to use these datasets for our defense, some preprocessing is required. The burst sequences varies in size between different visits and sites. However, the input to our models are fixed in size. In consequence, we must determine an ideal size of input and adjust our dataset to this size. To do this, we consider the distribution of burst sequence length within our datasets. Figure 3 shows the CDF of the burst sequence lengths for both half-duplex and full-duplex datasets. The figure shows that more than 80% of traces have less than 750 bursts for half-duplex closed-world dataset, and more than 80% of the traces for the closed-world full duplex dataset has less than 500 bursts. We next applied our adversarial defense to the half-duplex and full-duplex datasets. Using input sizes of 750 bursts and 1500 bursts in half-duplex dataset provides 92% and 93% accuracies with a simple CNN model (with three convolutional layers followed by two fully-connected layers). In addition, the accuracy rates of the full-duplex dataset with 750 bursts and 1500 bursts are 91% and 93%. The differences between the performances between the full size burst length and its short form would thus appear to be negligible. To decrease the computational cost to generate adversarial examples, we use the input size of 750 bursts for both full-duplex and half-duplex datasets for our evaluations.

5.2 Experimental Method

In our evaluation, we break each datasets (full-duplex and half-duplex) into two non-overlapping sets: the Adv Set \( \mathcal{A} \) and the Detector Set \( \mathcal{D} \) (see Table 1). The Adv Set and Detector Set both contains half of the instances of each class. For full-duplex data, each set contains 259 instances from each 95 classes, while the half-duplex sets contains 83 classes each with 360 instances.

The WF defender needs to train a detector \( f(x) \) with instances of sites of his interest to generate adversarial traces. We use the CNN model in designed by Sirinam et al. [29] as our detector and train it on the traces of the Detector Set. The Detector Set is only used to train the detector. We select the source samples that we are generating their adversarial version from Adv Set \( (I_s \in \mathcal{A}) \).

Evaluation Scenarios

In our algorithm, we try to change the source samples toward a target sample drawn randomly from our target pool. The detector in our defense is responsible to verify whether
Table 1: Dataset Split: Adv Set & Detector Set.

|                     | Adv Set $\mathcal{A}$ | Detector Set $\mathcal{D}$ | Closed-World | Open-World |
|---------------------|------------------------|-----------------------------|--------------|------------|
|                     | (Class $\times$ Instances) | (Class $\times$ Instances) | Total        |        |
| Full-Duplex         | $95 \times 259$       | $95 \times 259$             | $95 \times 518$ | 40,716     |
| Half-Duplex         | $83 \times 360$       | $83 \times 360$             | $83 \times 720$ | 40,000     |

The perturbed source sample is still in the source class. We are interested to know how the algorithm performs if we fill the target pool with instances of sites that the detector has been trained on and has not been trained on. We examine the bandwidth overhead and reduction in attack accuracy of traces protected by our method in these two different scenarios.

- Case I: We fill the target pool with instances from the Adv Set. Therefore, both source samples ($I_s \in \mathcal{A}$) and target samples ($I^T \in \mathcal{A}$ which $T_i \neq s$) are from the Adv Set. In this case, we assume that the detector has been trained on the target classes.
- Case II: We fill the target pool with instances from the unmonitored sites. We select the target samples ($I^T \in \mathcal{D}$) from the open-world dataset. The source samples ($I_s$) are from Adv Set and we generate their defended forms. The Detector Set is only used to train the detector. In this case, we assume the detector has not been trained on the target samples.

We generate defended samples with various settings. We vary $\alpha$ to evaluate their effect on the strength of the defended traces and the overhead. We measure the detectability of the defended samples by applying the DF attack [29] on them. Sirinam et al. [29] suggest using 5,000 packets. The WF defenses protect the traffic traces by adding dummy packets to the traffic, this increases the number of packets in the traces. Therefore, the number of packets in the defended traces are larger than their undefended forms. We thus increase the input size of the DF attack and use an input size of 10,000 packets, which is the 80th percentile of packet sequence lengths in our defended traces.

We also vary the number of iterations required to generate the adversarial traces. This iteration implies that the traces are being changed in each iteration. The bandwidth overhead increases with the increase of iteration. On the other hand, the accuracy gets better and better. However, we cannot make the changes to an unreasonable number of iterations. We want to control the bandwidth overhead to a limit with an acceptable attack accuracy.

5.3 Experiments with Full-Duplex Data

Alpha Value Figure 4 shows the bandwidth overhead and attack accuracy of full-duplex data with respect to $\alpha$ values for both Case I (solid lines) and Case II (dashed lines) when the number of iteration is 500. As shown in the figure, the bandwidth overhead increases and the attack accuracy decreases as we increase $\alpha$. Larger $\alpha$ values create longer steps toward the target samples lead to higher bandwidth overhead.

For Case I, the accuracy of $\alpha=5$ and $\alpha=7$ are the same and it is 35%, but the bandwidth overhead is lower for $\alpha=5$, it is 56% which is 3% lower compared to $\alpha=7$. For Case II, the accuracy and the bandwidth overhead are slightly lower for...
\( \alpha = 5 \) than that of \( \alpha = 7 \). From these findings, we fix \( \alpha = 5 \) for full-duplex data.

We can also observe that, Case I leads to lower accuracy and comparable amount of bandwidth overhead than that of Case II. Therefore, picking target samples from classes that the detector has been trained on will drop the accuracy.

**Number of Iterations** We can see the trade-off between the accuracy and bandwidth overhead with respect to the increase of the number of iterations to generate the adversarial traces (see Figure 5). As we mentioned earlier that increasing the number of iterations also increase the number of packets (overhead) in the defended traces. We vary the number of iterations from 100 to 500 for both Case I (Figure 6a) and Case II (Figure 6b) to see their impact on the overhead and the accuracy rate of the DF attack.

For Case I, we can see that the DF attack accuracy for 400 and 500 iterations are the same that is 35% when \( \alpha = 5 \). The attack accuracy is 3% higher for when \( \alpha = 7 \) for 400 iterations. The bandwidth overheads for \( \alpha = 5 \) with 400 and 500 iterations are 54% and 56%, respectively. For \( \alpha = 7 \), it is 2% higher for both 400 and 500 iterations. This is another evidence that \( \alpha = 5 \) is better to generate adversarial traces.

For Case II, \( \alpha = 5 \) provides us with 57% accuracy with 53% bandwidth overhead for 400 iterations. For 500 iterations, it provides 55% accuracy and 56% bandwidth overhead. As a defender we want to reduce the attack accuracy as much as possible with an acceptable amount of bandwidth overhead. From the figures, we can interpret that 500 iterations provides reduced accuracy than 400 iterations with comparable amount of bandwidth overhead. Hence, we keep the number of iterations to 500.

### 5.4 Experiments with Half-Duplex Data

**Alpha Value** According to Figure 5, the lowest accuracy rate is 35.5% and 28.8% for Case I and Case II, respectively, when \( \alpha = 7 \). The bandwidth overheads are 62.7% and 73.5% for Case I and Case II, respectively. When \( \alpha = 5 \), the attack accuracy is 50% for both Case I and Case II with bandwidth overheads 57% and 69%, respectively. As a defense, the reduced accuracy is more advisable even with a little more cost of bandwidth. Though \( \alpha = 5 \) provides moderate reduced bandwidth overhead, the attack accuracy are higher for both cases compared to \( \alpha = 7 \). Hence, for the half-duplex data, \( \alpha = 7 \) is the preferable choice. All of the results shown in Figure 5 are based on 500 iterations.

We can also notice that Case I provides lower bandwidth overhead and lower detectability than Case II. This means that picking target samples from classes that the detector trained on reduces both attack accuracy and bandwidth overhead. For Case II, \( \alpha = 5 \) provides us with 57% accuracy with 53% bandwidth overhead for 400 iterations. For 500 iterations, it provides 55.5% accuracy and 56.5% bandwidth overhead. As a defender we want to reduce the attack accuracy as much as possible with an acceptable amount of bandwidth overhead. We can understand from the graphs and the explanations that iterations=500 provides reduced accuracy than iterations=400 with comparable amount of bandwidth overhead. Hence, we keep the number of iterations to 500.

**Number of Iterations** Figure 7 [shows](#) the trade-off between the accuracy and bandwidth overhead with the increase of the number of iterations to generate the adversarial traces, when \( \alpha = 7 \). We vary the number of iterations from 100 to 500 for both Case I and Case II. For Case I, the accuracy is 35.5% with 62.7% bandwidth overhead with 500 iterations. With 400 iterations, the accuracy is 37% with 59% bandwidth overhead. For Case II, with 500 iterations, we can get the lowest attack accuracy among all the settings in our defense which is 28.8%, with a cost of 73.5% bandwidth overhead. We can observe that the bandwidth overhead for Case I is always lower than Case II.

### 5.5 Results Analysis

We have presented a comparison of attack accuracy and bandwidth overhead of Adv-DWF against DF attack and CU-MUL attack for both Case I and Case II [in Table 2](#). We also show the comparison of the performance of Adv-DWF with two state-of-the-art defenses WTF-PAD and Walkie-Talkie (W-T). In Table 3, we show the comparison of Top-2 accuracy of our defense and W-T defense.

**Bandwidth Overhead**

As explained in Section 4.4, we designed Adv-DWF algorithm to minimize packet addition by keeping the amount of change in a trace by a threshold value of 0.0001. Keeping the amount of change at a minimum keeps the bandwidth overhead as low as possible. We show comparison against WTF-PAD and W-T because these are the two lightweight and state-of-the-art defenses. As mentioned earlier the other proposed WF defenses have 2-3 times higher bandwidth overhead. Thus, we only consider WTF-PAD and W-T for a fair comparison.

From Table 2, we can see that, with full-duplex data, the bandwidth overhead in Case I and Case II of Adv-DWF are the same 56.5% and at least 7% and 12% lower than WTF-PAD and W-T defense, respectively. In contrast, the bandwidth overhead for half-duplex data is lower only for Case I than WTF-PAD and W-T.

The Case II for half-duplex network traffic requires at least 10% higher bandwidth overhead than Case I of half-duplex. In terms of bandwidth overhead, Case I is better for both full-duplex and half-duplex network traffic, although not to a significant degree.

In summary, our bandwidth overheads are better than WTF-PAD and W-T in most of the cases, except one. This
Bandwidth Overhead

Figure 6: **Full-duplex - Accuracy & Bandwidth Overhead**: the accuracies & bandwidth overhead of the generated samples against the DF attack with respect to the variation of the number of iterations to generate the adversarial traces. (a) represents the Case I and (b) represents the Case II, of our evaluation scenarios.

Table 2: The Evaluation of the Adversarial Traces against the DF and CUMUL Attacks & Comparison against WTF-PAD and W-T Defenses.

| Cases        | Dataset                                                                 | BW Overhead (%) | DF [29] (%) | CUMUL [21] (%) |
|--------------|-------------------------------------------------------------------------|-----------------|-------------|----------------|
| Undefended   |                                                                         | -               | 0.98        | 0.973          |
| WTF-PAD [14] |                                                                         | 0.640           | 0.907       | 0.603          |
| Walkie-Talkie (W-T) [33] |                                                               | 0.690           | 0.497       | 0.384          |
| Case I       | Adversarial Traces (Full-duplex)                                       | **0.565**       | **0.352**   | **0.232**      |
|              | Adversarial Traces (Half-duplex)                                       | 0.627           | 0.355       | 0.301          |
| Case II      | Adversarial Traces (Full-duplex)                                       | **0.565**       | 0.555       | **0.368**      |
|              | Adversarial Traces (Half-duplex)                                       | 0.735           | **0.288**   | **0.316**      |

Table 3: Top-2 Accuracy of DF Attack against Adversarial Traces and W-T.

| Cases        | Dataset                                                                 | DF [29] Top-2 Accuracy (%) |
|--------------|-------------------------------------------------------------------------|----------------------------|
| Walkie-Talkie (W-T) [33] |                                                                  | 0.984                      |
| Case I       | Adversarial Traces (Full-duplex)                                       | **0.514**                  |
|              | Adversarial Traces (Half-duplex)                                       | **0.528**                  |
| Case II      | Adversarial Traces (Full-duplex)                                       | **0.569**                  |
|              | Adversarial Traces (Half-duplex)                                       | **0.432**                  |

amount of bandwidth overhead can be acceptable in comparison to other WF defenses for Tor.

**Attack Accuracy**

Interestingly, we observe that the accuracies of the DF attack in Case I for both full-duplex and half-duplex network traffic are 35.2% and 35.5%, respectively, indicating that DF attack fails against our defense. In addition, the DF attack accuracies are significantly lower than WTF-PAD and W-T. More precisely, we can see from Table 2 that accuracies are at least 45% and 14% lower than that of WTF-PAD and W-T.

In Case II, the accuracy of the DF attack against full-duplex network traffic is 55.5% which is slightly higher than W-T but 45% lower than WTF-PAD. Surprisingly, the half-duplex adversarial traces provide the best performance against the DF attack and can drastically drop the accuracy down to 28.8%, showing that DF attack also fails against Case II of our defense.

The CUMUL attack can achieve at best 36.8% accuracy against full-duplex network traffic of Case I. In all the other cases, the CUMUL attack accuracies against our defense are closer to 30% or lower than that. We also want to point out that, CUMUL attack accuracies against our defense are lower than that of WTF-PAD and W-T as well (see Table 2). It is important to note that the CUMUL attack is also unsuccessful against our defense. This results show that the effectiveness of Adv-DWF is not limited to only against the deep-learning based DF attack, but also traditional machine-learning based CUMUL attack.

It is also notable that Case I provides better consisten-
cies between the attack accuracies for both full-duplex and half-duplex network traffic in both DF and CUMUL attacks. Overall, the results demonstrate that Adv-DWF is effective and has significantly lower attack performance compared to the state-of-the-art WF defenses, WTF-PAD and W-T.

**Top-2 Accuracy**

In general, prior works have focused their analysis on the Top-1 accuracy, which is normally referred to simply as the accuracy of the attack. The Top-1 accuracy, however, does not provide a full picture of the effectiveness of a defense as it ignores any additional classes which may have been a strong candidate for the final classification. An attacker may use additional insights about their target (language, locale, interests, etc) to further deduce what website their target is likely to visit. Furthermore, most WF defenses are evaluated in a simulated environment like our own. These simulators abstract away networking issues that may occur when the defense is implemented and limited by real-world network conditions. These real-world limitations can produce imperfections when applying the defense and may expose the user to increased fingerprintability.

As such, it is desirable to examine the accuracy of Top-$N$ predictions, where $N$ is the number of top-ranked classes in the prediction. In evaluations of WF, we are particularly interested in the Top-2 accuracy. A high Top-2 accuracy indicates that the classifier is able to reliably determine the identity of a trace to one of two candidate websites. This is a threat to a user even when the attacker is unable to use additional information to predict the true site. The knowledge that the target may be visiting a sensitive site is actionable and can encourage the attacker to expend additional resources to further analyze their target.

To better evaluate the efficacy of Adv-DWF against this threat, we compute the Top-2 accuracy and compare it to that of W-T. The results show that Adv-DWF is resistant to Top-2 identification, with an accuracy of 56.9% in the worst case. On the other hand, W-T struggles in this scenario as its defense design does not protect beyond confusion between two classes. This means that Adv-DWF users are at notably lower risk of de-anonymization. Thus, Adv-DWF remains a strong defense against WF attacks in all threat scenarios.

## 6 Conclusion

We propose a novel WF defense, Adv-DWF, that can fool the state-of-the-art WF attack with a significant amount of reduced attack accuracy. Our defense has lower bandwidth overhead than WTF-PAD and Walkie-Talkie, the state-of-the-art defenses. The defense uses a novel mechanism that adapts techniques to create adversarial traces against machine-learning and deep-learning classifiers. The generated adversarial traces can significantly limit the ability of a WF adversary to distinguish a site, even though his classifier is trained on the adversarial traces. Our defense mechanism results in 56% bandwidth overhead and drops the accuracy of the state-of-the-art WF attack from 98% to 35%. We successfully show that our defense can fool the deep-learning-based attack. In addition, our defense performs better than the two state-of-the-art defenses: WTF-PAD and W-T. The Top-2 attack accuracy of our defense in DF attack is at best 56.9%, whereas it is 98.4% for W-T. We emphasize that our experiments are conducted in the closed-world setting, where the attacker knows that the Tor client is assumed to visit one of the monitored sites. In a more realistic open-world setting, where the client could visit any site on the Web, 35% accuracy is very likely to lead to many false positives for the attacker. Overall, the results of our defense indicate that it can be a potential defense for Tor against WF attacks.

### Availability

The code and datasets will be released upon the publication of this paper.

### References

[1] Alexa. [http://www.alexa.com](http://www.alexa.com).

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

[3] Cai, X., Nithyanand, R., and Johnson, R. CS-BuFLO: A Congestion Sensitive Website Fingerprint-
ing Defense. In Workshop on Privacy in the Electronic Society (WPES) (2014), ACM.

[4] CAI, X., NITHYANAND, R., WANG, T., JOHNSON, R., AND GOLDBERG, I. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In ACM Conference on Computer and Communications Security (CCS) (2014).

[5] CAI, X., ZHANG, X. C., JOSHI, B., AND JOHNSON, R. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In ACM Conference on Computer and Communications Security (CCS) (2012).

[6] CARLINI, N., AND WAGNER, D. A. Adversarial examples are not easily detected: Bypassing ten detection methods. In AISec@ CCS (2017).

[7] CHENG, H., , CHENG, H., AND AVNUR, R. Traffic analysis of ssl encrypted web browsing, 1998.

[8] CHERUBIN, G., HAYES, J., AND JUAREZ, M. "Website fingerprinting defenses at the application layer". In Privacy Enhancing Technologies Symposium (PETS) (2017), De Gruyter.

[9] DYER, K. P., COULL, S. E., RISTENPART, T., AND SHRIMPTON, T. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In S&P (2012).

[10] GOODFELLOW, I. J., SHILENS, J., AND SZEGEDY, C. Explaining and harnessing adversarial examples, 2014.

[11] HAYES, J., AND DANEZIS, G. k-fingerprinting: A robust scalable website fingerprinting technique. In USENIX Security Symposium (2016), USENIX Association.

[12] HERRMANN, D., WENDOLSKY, R., AND FEDER-RATH, H. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In ACM workshop on Cloud computing security (2009), ACM.

[13] JUAREZ, M., AFROZ, S., ACAR, G., DIAZ, C., AND GREENSTADT, R. A critical evaluation of website fingerprinting attacks. In ACM Conference on Computer and Communications Security (CCS) (2014).

[14] JUAREZ, M., IMANI, M., PERRY, M., DIAZ, C., AND WRIGHT, M. Toward an efficient website fingerprinting defense. In European Symposium on Research in Computer Security (ESORICS) (2016).

[15] KOWN, A., ALSABAH, M., LAZAR, D., DACIER, M., AND DEVADAS, S. Circuit fingerprinting attacks: Passive deanonymization of Tor hidden services. In USENIX Security Symposium (2015), USENIX Association.

[16] LIU, Y., CHEN, X., LIU, C., AND SONG, D. Delving into transferable adversarial examples and black-box attacks, 2016.

[17] LUO, X., ZHOU, P., CHAN, E., AND LEE, W. HTTPoS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. In Network & Distributed System Security Symposium (NDSS) (2011).

[18] NITHYANAND, R., CAI, X., AND JOHNSON, R. Glove: A Bespoke Website Fingerprinting Defense. In Workshop on Privacy in the Electronic Society (WPES) (2014), ACM.

[19] NITHYANAND, R., CAI, X., AND JOHNSON, R. Glove: A bespoke website fingerprinting defense. In Workshop on Privacy in the Electronic Society (WPES) (2014), ACM.

[20] OVERDORF, R., JUAREZ, M., ACAR, G., GREENSTADT, R., AND DIAZ, C. How unique is your .onion? an analysis of the fingerprintability of Tor onion services. In ACM Conference on Computer and Communications Security (CCS) (2017).

[21] PANCHEKHO, A., LANZE, F., PENNEKAMP, J., ENGEL, T., ZINNEN, A., HENZE, M., AND WEHRLE, K. Website fingerprinting at internet scale. In Network & Distributed System Security Symposium (NDSS) (2016).

[22] PAPERNOT, N., McDaniel, P., GoDfellow, I., Jha, S., Celik, Z. B., AND SWAMI, A. Practical black-box attacks against machine learning. In ACM Asia Conference on Computer and Communications Security (2017).

[23] PAPERNOT, N., McDaniel, P., Jha, S., FREDRIKSON, M., Celik, Z. B., AND SWAMI, A. The limitations of deep learning in adversarial settings, 2015.

[24] PAPERNOT, N., McDaniel, P., Wu, X., Jha, S., AND SWAMI, A. Distillation as a defense to adversarial perturbations against deep neural networks, 2015.

[25] PERRY, M. Experimental Defense for Website Traffic Fingerprinting. Tor Project Blog, 2011. https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting

[26] PERRY, M. Padding negotiation. Tor Protocol Specification Proposal. https://gitweb.torproject.org/torspec.git/tree/proposals/254-padding-negotiation.txt, 2015. (accessed: February 15, 2019).
A Scenarios of Existing Methods

In order to evaluate the efficacy of the adversarial traces generated by the Carlini method described in Section 4.3, we consider two different attack scenarios based on when the defenses are applied. The defense can be applied either after the attack or before the attack. We evaluate both scenarios in the following sections.

A.1 SCENARIO I: Defense After Attack

In this scenario, we assume that the attacker has trained its classifier on the traces that have not been defended. We assume that the attacker is not aware of a defense in place. Such a scenario can be valid in the case that we have an attacker that does not target any specific client and his goal is to identify the behavior of large number of users, but some of the clients may use some defense to protect their traffic. Here, the attacker is not aware of the defense or his goal is to monitor the majority of the clients.

For this scenario, we first train a classifier on our undefended traces and then we generate the adversarial traces. We examine the efficacy of the generated samples by testing them against the trained attacker model.

In our evaluation we break the data into two sets, Adv Set, and Detector Set, each set has 83 classes with 360 instances each. The attacker trains the classifier on the Detector Set. The traces in Detector Set are not protected by any defenses. The WF attacks that we apply on the Adv Set are the DF and CUMUL attacks. We chose DF attack as a state-of-the-art WF attack representing Deep Learning WF attacks, and CUMUL attack as the representative of the traditional machine learning algorithms. CUMUL attack uses SVM classifier and has high performance compared to other tradition WF attacks.

We apply the method described in [4.3] to generate adversarial traces from the traces in Adv Set, we call these traces in our evaluation as Adversarial Traces. To generate Adversarial Traces, we use our Simple CNN as the target model (F) and the adversarial traces will be generated based on simple CNN. The architecture of our simple CNN is shown in Table 6.

Table 4 shows the results of our evaluations. According to the table, Adversarial Traces add 62% bandwidth overhead. Adversarial Traces generated for Simple CNN can confound the target model 98% of the times. In addition, the accuracy of DF and CUMUL attacks are 3% and 31%. This means that Adversarial Traces generated based on a target model with Simple CNN architecture, can be highly transferable to other machine learning models. Almost all the adversarial traces generated by Simple CNN can confound DF attack, which is also a deep learning model. The results show that the adversarial traces are more transferable to DNN model than traditional machine learning models.

A.2 SCENARIO II: Attack After Defense

In this scenario, we assume that the attacker knows that the client is using some sort of defense mechanisms to protect her traffic. The attacker then collects the traces protected by the same method as the client and trains his classifier with those traces. In this scenario, the training set and testing set are both traces protected by the same WF defense method. This scenario is more realistic because it has been shown that the effectiveness of the WF attacks depends on the attacker’s knowledge of the clients. Moreover, once a defense is
deployed it is supposed to be accessible for all the users and used by all the users. Therefore, the attacker can also use the same defense as other clients.

For evaluation in this scenario, we protect the traces in Adv Set by Adversarial Traces (described in Section 4.3) using a target model with the architecture in Table 6. Then we train the WF attacks, Simple CNN, DF, and CUMUL attacks, on 90% defended traces in Defender Set and test them with the remaining 10% of defended traces.

To generate Adversarial Traces, we train a target model with the same architecture as Simple CNN with the traces in Detector Set and used in generating Adversarial Traces on Adv Set. The results of the evaluation in this scenario are shown in Table 5. As shown in the table, even when adversarial traces are generated based on a target model with similar architecture as Simple CNN, they are highly detectable on Simple CNN as we train Simple CNN on the adversarial traces, and its accuracy is 91%. Moreover, DF and CUMUL attacks can also detect the adversarial traces with high accuracy rate, 97% and 91%, respectively. This means that generated adversarial traces are ineffective as the attacker is trained on the adversarial traces. Generating the adversarial traces works like a data augmentation technique in this case. If the attacker is trained on them, the attacker will detect them correctly. This highlights the necessity of the creation of a new adversarial example generation technique designed specifically for WF.