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

Mohsen Imani*
The University of Texas at Arlington
mohsen.imani@mavs.uta.edu

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

Nate Matthews
Center for Cybersecurity
Rochester Institute of Technology
nate.mathews@mail.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. two-to-three times as large or slow) or ineffective against the latest attacks. In this paper, we explore a novel defense, *Mockingbird*, based on the idea of *adversarial examples* that have been shown to undermine machine learning classifiers in other domains. To make the technique more robust than existing algorithms for finding adversarial examples, our approach aims to make the source trace more similar to a randomly selected target and use the neural network only to determine when the modified trace will be misclassified with high confidence. The technique drops the accuracy of the state-of-the-art attack hardened with adversarial training from 95% to between 29-57%, depending on the scenario, while incurring a reasonable 56% bandwidth overhead. The attack accuracy is generally lower than state-of-the-art defenses, and much lower when considering Top-2 accuracy, while incurring lower overheads in most settings. In addition, the information leakage of *Mockingbird* is at most 1.9 bits, whereas it is at most 2.2 bits for W-T and 2.6 bits for WTF-PAD. .

KEYWORDS

Anonymity System; Defense; Privacy; Website Fingerprinting; Adversarial Machine Learning; Deep Learning;

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 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 hidden patterns in a dataset. The DF attack can successfully reach up to 98% accuracy to distinguish a site in an undefended dataset in a *closed-world* setting [36], 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 [4, 5, 18, 22, 25, 31, 40, 41]. 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 [6], CS-BuFLO [4], and Tamaraw [5]) apply these techniques effectively, but require high overheads and make loading a website take two or three times as long as in Tor. Recently, two relatively lightweight defenses have been proposed, WTF-PAD and Walkie-Talkie (W-T). WTF-PAD [18] offers lower overheads and is considered to be a leading candidate for deployment in Tor [32]. Sirinam et al., however, 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 [40] (W-T) performs better, but the DF attack can still attain a Top-2 accuracy of 98%, which provides an attacker the ability to reliably scope down what the user is visiting to just two sites [36].

In this paper, we introduce *Mockingbird*,* a new defense strategy using adversarial examples generated by a deep neural network. There are various techniques to generate these examples in the context of image classification [8, 24, 28, 29, 37], but most of them perform a search in the space of possible images. Starting with the original image, the search moves gradually towards a slightly modified image that is classified differently from the source. This search is typically guided by the gradient of the neural network's loss function or the *logits*, the inputs to the neural network's final classification decision. Adversarial examples generated by these methods have had tremendous success in fooling deep-learning

***The authors contribute equally to this paper.**

*The Northern mockingbird imperfectly but effectively imitates the calls of a wide range of other birds, and one of its call types is known as a *chatburst* that it uses for territorial defense: https://en.wikipedia.org/wiki/Northern_mockingbird.

models to misclassify with as much as 100% accuracy and low amounts of distortion in the images [9]. This makes adversarial examples intriguing for defending against WF attacks using deep learning, as they could fool the classifier without adding large amounts of distortion in the form of padding or delay.

While neural networks can be made more robust against adversarial examples, most of these techniques are not effective upon careful analysis [3, 7] or do not generalize to new approaches to generating the examples [33]. In the WF context, however, the defense is generating the adversarial traces, so the attacker has the advantage of going second. In particular, he can use the defense as an oracle to generate these traces and use them to train his classifier to be more robust. This *adversarial training* approach has been shown to be effective when the classifier knows how the adversarial traces are being generated [19].

To address this, we propose a novel technique to generate adversarial traces that limits the influence of the design and training of the neural network in finding a good adversarial example. In particular, as we search for the new trace, we gradually reduce the distance to a randomly selected nearby trace. The neural network is only used to give a confidence value on whether the current trace fools the classifier, and we do not access the loss function, the logits, or any other aspect of the neural network. The resulting adversarial traces can go in many different directions from the original source traces instead of consistently following the same paths that result from, e.g., following the gradient of the loss function. We also add constraints to this approach to make sure the outputs are suitable for network traces.

With this technique, *Mockingbird* reliably causes misclassification in a deep-learning classifier hardened with adversarial training using only moderate amounts of bandwidth overhead.

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 show how the state-of-the-art algorithms in generating adversarial examples in computer vision fail as a defense in the WF setting.
- We describe a novel algorithm, *Mockingbird*, to generate adversarial traces that can fool a deep-learning classifier hardened with adversarial training. The algorithm is more robust than other approaches because it is more random in its search method and relies less on the shape of the underlying neural network. Additionally, *Mockingbird* is specifically designed to account for constraints in the WF problem to generate suitable network traces.
- We evaluate *Mockingbird* and find that it significantly reduces accuracy of the state-of-the-art WF attack hardened with adversarial training from 98% to 29%-57%, depending on the scenario. The bandwidth overhead is 57% for full-duplex traffic with no added delays, which is better than both W-T and WTF-PAD.
- We show that it is harder for an attacker to scope down to just two sites against our defense. The state-of-the-art attack can reach at most 57% Top-2 accuracy against our defense, while the Top-2 accuracy on W-T and WTF-PAD is 97% and 90%, respectively.

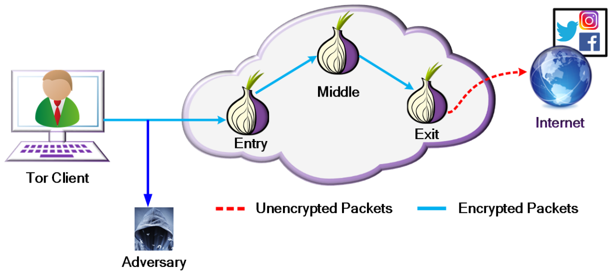


Figure 1: WF Attack Model.

- Using the WeFDE framework [20], we measure the information leakage of our defense with a large set of features. We find that *Mockingbird* has less leakage for many types of features than either W-T or WTF-PAD, which confirms our findings for accuracy and Top-2 accuracy.

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 the 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 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 potentially 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 a traffic trace of one site indistinguishable from traces of other sites. To achieve this, the real traffic stream must be manipulated in some way. Because 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. However, the guard node has knowledge about the identity of the client and is in a position to act as a WF adversary. Therefore, it would be ideal to set up the bridge at the middle node, which cannot directly identify the client.

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 leaks in encrypted HTTP requests that could expose the identity of the site being visited [10, 38]. The threat of website fingerprinting against Tor was first evaluated by Herrmann et al. [16] 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 attack's failure against Tor, with 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 that utilize hand-crafted features as well as recent deep-learning based attacks.

k-NN. The k-NN attack introduced by Wang et al. [39], uses a modified k-nearest neighbors (k-NN) classifier. 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. [27] 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. [36] shows that CUMUL attack has comparable performance with the DF attack. Thus, 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 [15], who 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 uses 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, and these 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 direction, represented by a sequence of "+1" and "-1" values for outgoing and incoming packets, respectively. Despite this innovation, their attack achieved a lower accuracy rate than the previous state-of-the-art attacks at only 88% in the closed-world setting. The reason for 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. [34] 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, obtaining 96% accuracy in the closed-world setting.

Deep Fingerprinting (DF). Sirinam et al. [36] 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 concerningly high performance against even some defended traffic. Most notably, their attack achieved 90% accuracy against WTF-PAD [18], the most likely WF defense to be deployed on Tor, and 98% Top-2 accuracy against Walkie-Talkie [40].

[†]Tor bridges are usually used for evading censorship, but they can be used for prototyping WF defenses such as used in WTF-PAD [18].

3.2 WF Defenses

In an effort to defeat WF attackers, researchers have explored various defense designs that generate cover traffic to hide the features present in website traffic. WF defenses are able to manipulate the traffic stream with two operations: sending dummy packets and delaying real packets. These manipulations, however, come at a cost: sending dummy packets adds an additional bandwidth overhead to the network, while delaying packets adds latency overhead that 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 [13] 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 course-grained features like the volume and load time of the page still leak information about the website. Tamaraw [5] and CS-BuFLO [4] extend the BuFLO design with the goal of addressing these issues. Since nearly all sites send more traffic to the client than the client sends to them, 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, requiring two to three times as much time as Tor to fetch a typical site 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 [39], Glove [26], and Walkie-Talkie [40]. 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 sizes of just two, this produces a theoretical maximum accuracy of 50% . WT reduces the browser to a *half-duplex* mode of operation, in which the browser sends new requests only after the responses to all the previous requests have been received. WT seeks to pair sensitive websites with nonsensitive websites, and due to WT's theoretical maximum attacker accuracy of 50%, an attacker is unable to reliably determine whether a webpage visit was sensitive or not. Wang and Goldberg report just 31% bandwidth overhead for their defense, but also 34% latency overhead due to the use of half-duplex communication. Then Sirinam et al. [36] achieved 49.7% attack accuracy against WT.

Adaptive Padding (AP). Shmatikov and Wang [35] proposed Adaptive Padding (AP) as a countermeasure against end-to-end traffic analysis. Juarez et al. [18] 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 (where "large" is determined probabilistically), 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. Sirinam et al. [36], however, show that their DF attack can achieve up to 90% accuracy against WTF-PAD in the closed-world setting.

Application-level defenses. Cherubin et al. [12] 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. Cherubin et al. argued their defense was particularly suited for Onion Sites, which are typically more interested in preserving user privacy 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 accuracy of the CUMUL attack from 56% to 33%. Their second defense, LLaMA, 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.

4 DEFENSE DESIGN

We now motivate and describe the design of the *Mockingbird* defense, starting with a brief background on adversarial examples and adversarial training, then showing the limitations of applying existing techniques for generating adversarial examples, and finally showing the *Mockingbird* design in detail.

4.1 Adversarial Examples & Adversarial Training

Adversarial Examples. Szegedy et al. [37] were the first to discover that otherwise accurate ML and DL image classification models could be given image inputs with slight perturbations that would be largely imperceptible to humans but completely misclassified by the models. These perturbed inputs are called are called *adversarial examples*, and they call into question the robustness of many of the advances being made in machine learning. State-of-the-art DL models can be fooled into misclassifying adversarial examples with surprisingly high confidence. For example, Papernot et al. [28] show that adversarial images can cause a targeted deep neural network to misclassify 84% of the time.

The idea 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 original class of x and the target class t 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 called a *targeted* adversarial example since it is misclassified to a particular target label t. However, if an adversarial example $x^{'}$ may be misclassified to any other classes except the true class ($C^*(x)$), it is called an *untargeted* adversarial example.

In response to the adversarial examples, many techniques have been introduced to make classifiers more robust against being fooled. Recent research [3, 9] shows that almost none of these recent approaches are effective. In particular, we can generate adversarial examples that counter these techniques by including the techniques directly into the optimization algorithm used to create the adversarial examples. We can also overcome many approaches by simply increasing the amount of perturbation used [7].

Adversarial Training. Recent research shows that adversarial training increases the robustness of a model by incorporating adversarial examples in the training data [19, 23]. The idea is to train a network or a classifier with adversarial examples so that they can be classified correctly. This approach is limited, as it does not adapt well to techniques for generating adversarial examples that haven't been trained on. In the WF setting, however, the classifier has the advantage of knowing how the adversarial examples are being generated, as they would be part of the open-source Tor code. Thus, adversarial training is a significant concern for our system.

4.2 Adversarial Examples as a WF Denfense

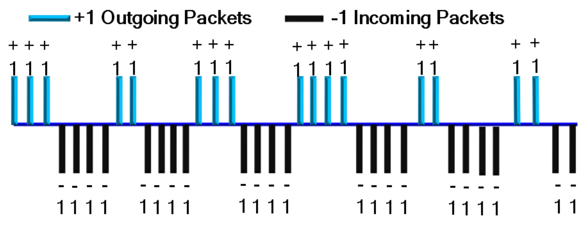


Figure 2: A visualization of bursts, with six outgoing bursts interspersed with six incoming bursts.

As we mentioned in above section, adversarial examples are generated from the distribution of correctly classified samples with slight perturbations. Moreover, adversarial examples have decent resistance against the existing defenses and they can remain undetectable. These two properties are compelling for developing a WF defense. In addition, 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.

From the perspective of developing a WF defense, the goal is to insert dummy packets to the original network traffic to fool the trained WF classifier. The number of dummy packets in the traffic should be as small as possible to keep bandwidth consumption low. These two goals of WF defenses, high rates of misclassification and low overhead, correspond to the two compelling properties of adversarial examples, which are to fool the classifier with transferablity using relatively small perturbations. Therefore, we propose to use the method of generating adversarial examples as a method of WF defense.

To develop a WF defense based on the adversarial examples, we first need to create a representation for the traffic traces that enables us to apply the adversarial examples algorithms on them. We model the traffic trace as a sequence of incoming (server to client) and outgoing (client to server) bursts. We define a burst as a sequence of consecutive packets in the same direction (see Figure 2). Given this, we can increase the length of any burst by sending padding packets in the direction of the burst. We cannot, however, decrease the size of the bursts by dropping real packets due to the retransmissions this would cause, changing the traffic patterns and adding delays for the user.

4.3 Existing Methods as a WF Defense

Several different algorithms have been proposed for generating adversarial examples within the field of computer vision, including the Fast Gradient Sign Method (FGSM) [14], the Jacobian-Based Saliency Map Attack (JSMA) [29], and optimization-based methods [9, 21]. For our initial exploration of adversarial examples in WF defense, we examine the technique proposed by Carlini and Wagner.

In 2017, Carlini and Wagner proposed a powerful optimization-based algorithm to generate adversarial examples [8]. This method was shown to defeat the defensive distillation approach of blocking adversarial examples [30]. The algorithm is successful with 100% probability. We modified their technique to suit our needs to generate adversarial traces out of the burst sequences. This algorithm is designed to work on images, which are 2D, so we modified it to work on the 1D traffic traces.

Given a sample x and a model F, the algorithm finds a perturbation δ that makes $x' = x + \delta$ to be misclassified to any other class than $C(x)$ ($C(x) = argmax_i F(x)_i$), or in the case of a 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 as L_0, L_2, or L_∞. The algorithm is formulated as follows:

$$\begin{aligned} \min \quad & \parallel \delta \parallel_p + c.f(x+\delta) \\ \text{such that} \quad & x+\delta \in [0,1]^n \end{aligned} \tag{1}$$

The algorithm will find δ such that it minimizes the distance metric, which is l_p norm, and the objective function $f(x+\delta)$. c is a constant to scale both the distance metric and objective function in the same range. Carlini and Wagner [8] used binary search to find the proper value for c. They explored several objective functions and found two taht work the best. For a targeted attack scenarios with target class t, the best objective function is:

$$f(x^{'}) = \max_{i \neq t}(F(x^{'})_i) - F(x^{'})_t \tag{2}$$

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

$$f(x^{'}) = F(x')_y - \max_{i \neq y}(F(x^{'})_i) \tag{3}$$

We evaluated the performance of this technique in two different WF attack scenarios: *without-adversarial-training* and *with-adversarial-training*. The without-adversarial-training 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 generated the adversarial examples against a target model and tested them against the different WF attacks trained on the original traffic traces. We find that our adversarial traces are highly effective against the WF attacks. The accuracy of DF [36], the state-of-the-art deep-learning attack, is reduced from 98% to 3%, and the accuracy of CUMUL [27] drops from 92% to 31%. Our adversarial traces generated using this method are highly transferable, as we generated them against a target CNN model and they are effective against both DF and CUMUL. The full details of these evaluations can be found in Appendix A.1.

This scenario is not realistic, unfortunately, 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 with-adversarial-training evaluations. When evaluated under this scenario, Carlini and Wagner's technique fails completely with 97% attack accuracy. The full details of these evaluations can be found in Appendix A.2.

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 Mockingbird, a novel algorithm to generate adversarial traces that more reliably fool the classifier in the adversarial training setting. The ultimate goal is to generate untargeted adversarial traces that cause the classifier to label a traffic trace as coming from any other site except the original site, i.e. to generate an untargeted sample.

To defend a given trace, the *source sample*, *Mockingbird* first generates a set of potential target traces selected randomly from the traces of various sites other than the source site. It then randomly picks one of these traces as the *target sample* and gradually changes the source sample to get closer to the target sample. The process stops when a trained classifier called the *detector* determines that the class of the sample has changed. Note that it does not need to have changed to the target sample's class, as the goal is to generate an untargeted adversarial trace. The amount of change applied to the source sample governs the bandwidth overhead of *Mockingbird*, and as such should be minimized.

Unlike most other algorithms used to generate adversarial examples, *Mockingbird* does not focus on the loss function (like FGSM and IGSM) or logits (like Carlini-Wagner) of the detector network. Instead, it aims to move the source sample towards the target sample and only uses the detector network to estimate the confidence with which the trace is misclassified. This lessens the reliance on the shape of the detector network and helps to generate adversarial traces that are more robust against adversarial training. We next explain the *Mockingbird* algorithm in detail.

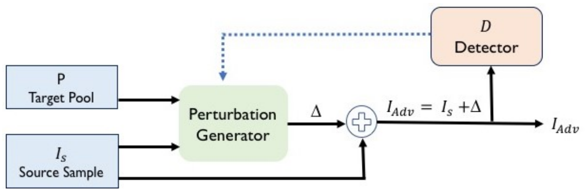


Figure 3: *Mockingbird* Architecture.

Mockingbird *Algorithm.* We assume that we have a set of sensitive sites $\mathcal{S}$ that we want to protect. We train a *detector* $f(x)$ on a set of data from $\mathcal{S}$. We discuss the design and training of $f(x)$ in Section 5. We consider traffic trace I_s as an instance of source class $s \in \mathcal{S}$. Our goal is to alter I_s such that it is classified to any other class t, $t = f(I_s)$ and $t \neq s$.

I_s is a sequence of the bursts, $I_s = \left[b_0^I, b_1^I, ..., b_n^I\right]$, where n is the length of the trace. The only allowed operation on a burst b_i^I is to add some positive values $\delta_i >= 0$ to that burst, $b_i^I = b_i^I + \delta_i$. The reason for using $\delta_i >= 0$ is that we want to only *increase* the volume of the bursts. If $\delta_i < 0$, 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 first select τ potential target samples from other classes $t_i \neq s$. We then select the target t as the one nearest to the source sample based on the l_2 norm distance. This helps to minimize overhead as we will move the source towards the target. More formally, we pick a *target pool* P_s of p random samples from other classes, $P_s = \left[I^0{}_0, I^1{}_1, .., I^p{}_m\right]$, where $I^j{}_i$ is the j-th sample in the target pool and belongs to target class $t_i \neq s$. The target sample I_t is selected as shown in Equation 4.

$$I_t = \underset{I \in P_s}{\operatorname{argmin}} D(I_s, I) \tag{4}$$

$$D(x, y) = l_2(x - y) \tag{5}$$

To make the source sample leave the source class, we change it with the minimum amount of perturbation in the direction that makes it closer to the target (I_t). We define Δ as the perturbation vector that we will add to the source sample to generate its defended form I_s^{new}.

$$\Delta = [\delta_0, \delta_1, \cdots, \delta_n] \quad (\delta_i >= 0) \tag{6}$$

$$I_s^{new} = I_s + \Delta \tag{7}$$

We need to find a Δ that adds the least amount of perturbation to the source sample while still making it closer to the target sample. Therefore, we find Δ that minimizes distance $D(I_s^{new}, I_T)$. To do so, we compute the gradient of the *distance* with respect to the input. Note that most work in adversarial example generation uses the gradient of the loss function of the discriminator network rather than distance, and this may make those techniques more sensitive to the design and training of the classifier. 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 modify the source sample in that direction towards the target sample. In particular:

$$\nabla(-D(I, I_T)) = -\frac{\partial D(I, I_T)}{\partial I} = \left[-\frac{\partial D(I, I_T)}{\partial b_i}\right]_{i \in [0, \cdots, n]} \tag{8}$$

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 $\nabla(-D(I, I_T))$ are positive. Our perturbation vector Δ is:

$$\delta_i = \begin{cases} -\alpha \times \frac{\partial D(I, I_T)}{\partial b_i} & -\frac{\partial D(I, I_T)}{\partial b_i} > 0 \\ 0 & -\frac{\partial D(I, I_T)}{\partial b_i} \leqslant 0 \end{cases} \tag{9}$$

where α is the parameter that amplifies the output of the gradient. The choice of α has an impact on the convergence and the bandwidth overhead. If we pick a large value for α, we will take bigger steps toward the target sample and add more overhead, while small values of α require more iterations to converge. We modify the source sample by summing it with Δ, ($I_s^{new} = I_s + \Delta$). We iterate this process, computing Δ for I_s and updating the source sample at each iteration until we leave the source class, $f(I_s^{new}) \neq s$ or the number of iterations passes the maximum allowed iterations. Note that at the end of each iteration, we update the current source sample with the modified one, $I_s = I_s^{new}$. 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 ($f_s(I_s^{new}) < \tau_c$), *Mockingbird* will stop changing the source sample (I_s).

As we only increase the size of the bursts where $-\frac{\partial D(I, I_T)}{\partial b_i} > 0$, we may run into cases that after some iterations $\nabla(-D(I, I_T))$ 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 $I_s^{new} - I_s$ is smaller than a threshold $\tau_D = 0.0001$ for λ consecutive iterations (we used $\lambda = 10$), and we are still in the source class, we select a new target. In particular, we effectively restart the algorithm by picking a new pool of potential target samples, selecting the nearest target from the pool, and continuing the process.

5 EVALUATION

5.1 Datasets

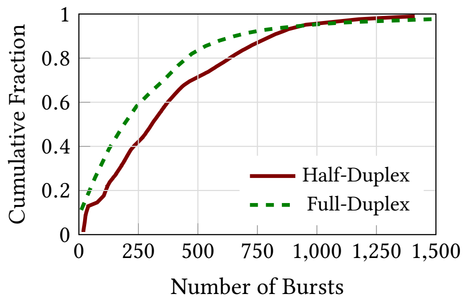


Figure 4: The CDF of the umber of bursts in the full duplex and half duplex traces.

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 (FD) and half-duplex (HD) communication modes. Walkie-Talkie (WT) [40] works on the half-duplex communication and it finds the supersequence in the burst level. In our experiments, we use burst sequences of both FD and HD datasets.

Full-Duplex (FD) : Sirinam et al. [36] collected a reasonably large dataset of the FD traffic traces. Their dataset has 95 classes with 1000 instances each. 95 sites are from the top 100 sites in Alexa [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 also use the FD open-world (OW) dataset from Sirinam et al. [36] in our experiments which has 40,716 different sites with 1 instance each. We process the OW dataset as well.

Half-Duplex (HD) : Sirinam et al. [36] 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 HD 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. Moreover, they also have OW data of 40,000 sites with 1 instance each. We also process OW dataset. We use both of these HD closed-world (CW) and OW datasets for our HD evaluations.

In order to use these datasets for our defense, some pre-processing is required before we can use those in our models. The burst sequences varies in size between different visits and sites. However, the input to our models should be 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 lengths within our datasets. Figure 4 shows the CDF of the burst sequence lengths for both HD and FD datasets. The figure shows that more than 80% of traces have less than 750 bursts for HD CW dataset, and more than 80% of the traces for the CW FD dataset has less than 500 bursts. We found that using input sizes of 750 bursts and 1500 bursts on both HD and HD datasets provides 1% improvement on accuracy on DF attack.The differences between the performances between the full size burst length and its short form (750 bursts) 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 FD and HD datasets in our evaluations.

5.2 Experimental Method

In our evaluation, we need to address the needs of both the *Mockingbird* system and the adversary to have training data. We thus break each dataset (full-duplex (FD) and half-duplex (HD)) into two non-overlapping sets: *Adv Set* $\mathcal{A}$ and *Detector Set* $\mathcal{D}$ (see Table 1). The Adv Set and Detector Set both contains half of the instances of each class. For FD data, each set contains 259 instances from each of 95 classes, while for HD data, each set contains 83 classes each with 360 instances.

Table 1: Dataset Split: *Adv Set* & *Detector Set*.

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

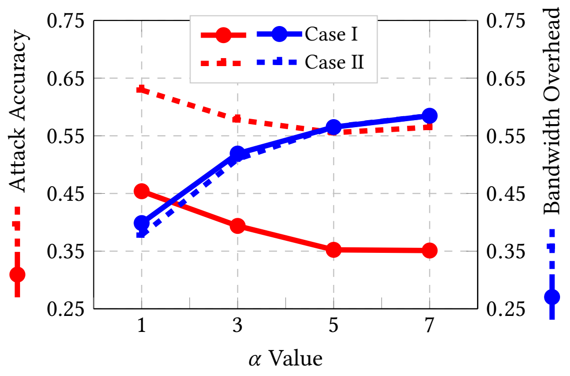


Figure 5: α *Value for full-duplex*: the accuracy and bandwidth overhead of generated samples as α varies. Dashed lines show the results of Case I and solid lines show the results of Case II.

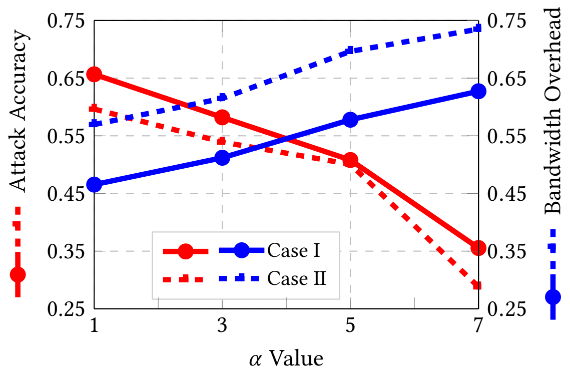


Figure 6: α *Value for half-duplex*: the accuracy and bandwidth overhead of generated samples as α varies. Dashed lines show Case I and solid lines show Case II.

Mockingbird needs to train a *detector* $f(x)$ with instances of a variety of sites to generate adversarial traces. We use the *Deep Fingerprinting (DF)* CNN model designed by Sirinam et al. [36] as our detector and train it on the traces of the Detector Set. Sirinam et al. suggest using an input size of 5,000 packets, but our padded traces have more traffic, so we use an input size of 10,000 packets, which is the 80th percentile of packet sequence lengths in our defended traces. The Detector Set is *only* used to train the detector. The source samples from Adv Set ($I_s \in \mathcal{A}$) are used to generate adversarial traces for the training, validation, and testing of the adversary's classifier.

We also use DF as the primary attack classifier for most of our experiments, as it is the state-of-the-art attack. Later, we show results using other attacks from the literature. In all of the evaluations in this section, the classifier is trained using adversarial training, where the attacker has full access to the defense and uses it to generate defended samples for each class in the monitored set.

Evaluation Scenarios. *Mockingbird* changes the source sample toward a target sample drawn randomly from our target pool. The *detector* is responsible to verify whether 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 the 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 (${I^j}_i \in \mathcal{A}$ 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, which makes it more effective at identifying when the sample has left one class for another. However, it may be less effective if the adversary trains on the source class but none of the other target classes used in detection.
- Case II: We fill the target pool with instances from unmonitored sites that are not in the Adv Set. We select the target samples (${I^j}_i$) from the open-world dataset. The source samples (I_s) are from Adv Set and we generate their defended forms. In this case, we assume the *detector* has not been trained on the target samples, so it may be less effective in identifying when the sample leaves the class. That may make it more robust when the attacker also trains on a different set of classes in his monitored set.

We generate defended samples with various settings. We vary α to evaluate their effect on the strength of the defended traces and the overhead. We also vary the number of iterations required to generate the adversarial traces. Each iteration moves the sample closer to the target, improving the likelihood it is misclassified, but also adds bandwidth overhead.

5.3 Experiments with Full-Duplex Data

The full-duplex version of *Mockingbird* is the easier to deploy and leads to lower latency costs than the half-duplex version [40], so we lead with the full-duplex results.

Choice of α Figure 5 shows the bandwidth overhead and attack accuracy of full-duplex data with respect to α values for both Case I (solid lines) and Case II (dashed lines) with 500 iterations. As expected, the bandwidth overhead increases and the attack accuracy decreases as we increase α, with longer steps towards the

Table 2: The Evaluation of the *Mockingbird* against DF, AWF, CUMUL, k-FP, and k-NN Attacks & Comparison against WTF-PAD and W-T Defenses. S×I: Sites×Instances, BWO: Bandwidth Overhead, FD: Full-Duplex, HD: Half-Duplex.

Cases	Dataset	S×I	BWO	DF [36]	AWF [34]	CUMUL [27]	k-FP [15]	k-NN [39]
	Undefended (FD)	95×259	-	0.95	0.88	0.93	0.85	0.86
	Undefended (HD)	83×360	-	0.98	0.93	0.92	0.92	0.90
	WTF-PAD [18]	95×463	0.640	0.85	0.39	0.55	0.44	0.17
	W-T [40]	82×360	0.720	0.362	0.39	0.36	0.30	0.35
Case I	*Mockingbird* (FD)	95×259	**0.565**	**0.352**	**0.20**	**0.232**	**0.21**	**0.04**
	Mockingbird (HD)	83×360	**0.627**	**0.355**	**0.27**	**0.300**	0.33	**0.06**
Case II	*Mockingbird* (FD)	95×259	**0.565**	0.555	**0.34**	0.368	0.34	**0.06**
	Mockingbird (HD)	83×360	0.735	**0.288**	**0.22**	**0.316**	0.30	**0.05**

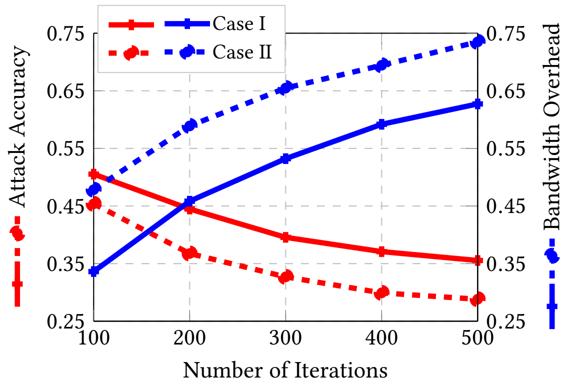


Figure 7: *Half-duplex*: the attacker's accuracy and the bandwidth overhead of the generated samples as we vary the number of iterations.

Table 3: Top-2 Accuracy of DF Attack against *Mockingbird* and W-T. FD: Full-duplex, HD: Half-Duplex.

Cases	Dataset	DF [36] Top-2 Accuracy
	WTF-PAD [18]	0.897
	W-T [40]	0.970
Case I	*Mockingbird* (FD)	**0.514**
	Mockingbird (HD)	**0.528**
Case II	*Mockingbird* (FD)	**0.569**
	Mockingbird (HD)	**0.432**

selected targets. For Case I, the adversary's accuracy against *Mockingbird* with α=5 and α=7 are both 35%, but the bandwidth overhead is lower for $\alpha = 5$ at 56% compared to 59% for $\alpha = 7$. For Case II, the adversary's accuracy and the bandwidth overhead are both slightly lower for α=5 than that of α=7.

As expected, we also observe that Case I leads to lower accuracy and comparable bandwidth overhead to Case II. When α=5 and α=7, the attack accuracies for Case I are at least 20% lower than that of Case II. Therefore, picking target samples from classes that the *detector* has been trained on drops the attacker's accuracy.

Number of Iterations. Figure 8 shows the trade-off between the accuracy and bandwidth overhead with respect to the number of iterations to generate the adversarial traces. As mentioned earlier, increasing the number of iterations also increases the number of packets (overhead) in the defended traces. We vary the number of iterations from 100 to 500 for both Case I (Figure 8a) and Case II (Figure 8b) to see their impact on the overhead and the accuracy rate of the DF attack.

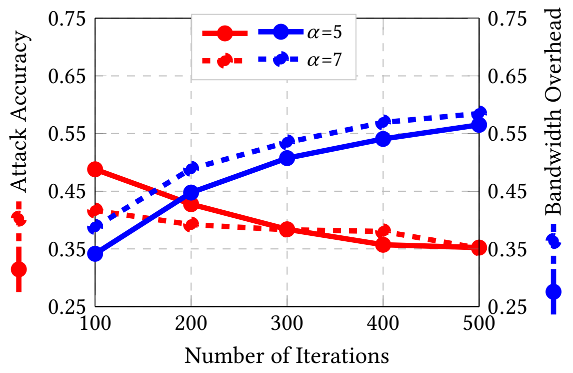
For Case I, we can see that the DF attack accuracy for both 400 and 500 iterations is 35% when α=5, while the bandwidth overheads are 54% and 56%, respectively. For $\alpha = 7$, the attacker's accuracy is higher and the bandwidth costs are higher. For Case II, using α=5 leads to 57% accuracy with 53% bandwidth overhead for 400 iterations and 55% accuracy and 56% bandwidth overhead for 500 iterations.

5.4 Experiments with Half-Duplex Data

Using half-duplex communication increases the complexity of deployment and adds latency overhead [40], but it offers more precise control over burst patterns such that we can achieve reduced attacker accuracy as shown in the following results.

Choice of α. As seen in Figure 6 (all for 500 iterations), the lowest accuracy rates are 35.5% and 28.8% for Case I and Case II, respectively, when α=7. The bandwidth overheads are 62.7% and 73.5% for Case I and Case II, respectively. When α=5, the attack accuracy is 50% for both Case I and Case II with bandwidth overheads of 57% and 69%, respectively. As expected, higher α values mean lower attacker accuracies at the cost of higher bandwidth. Additionally, we note that as in the full-duplex setting, Case I provides lower bandwidth overhead and lower detectability than Case II.

Number of Iterations. Figure 7 shows the trade-off between attacker accuracy and bandwidth overhead with the number of iterations for α=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.



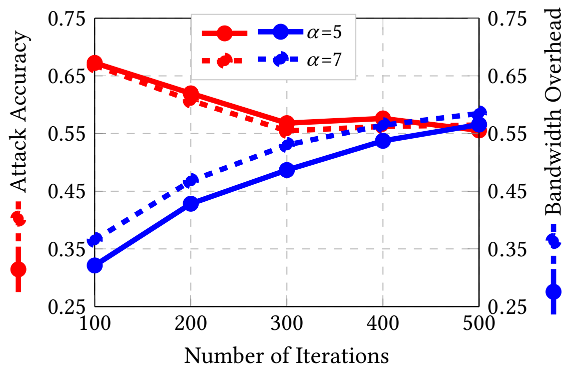


Figure 8: *Full-duplex*: the attacker's accuracy and the bandwidth overhead of the generated samples as we vary the number of iterations.

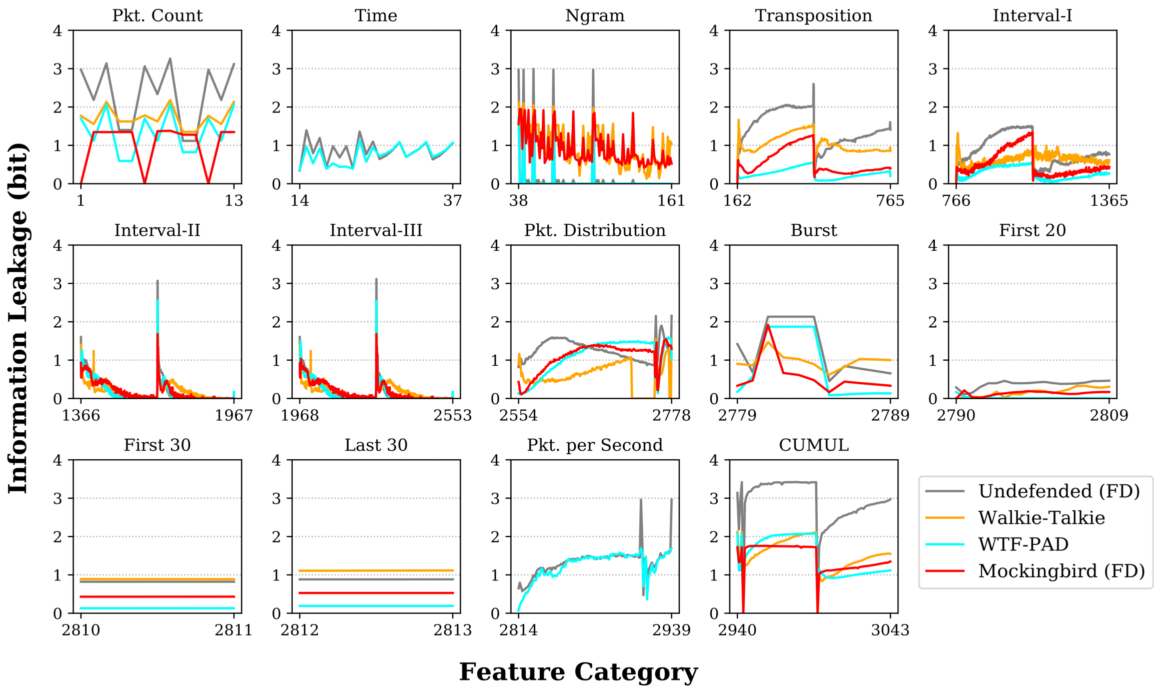


Figure 9: Individual feature information leakage

5.5 Information Leakage Analysis

Recent works have shown that classification accuracy is not a complete metric to evaluate the objective efficacy of a WF defense [11, 20]. To address this issue, we have adopted the WeFDE information leakage estimation technique [20] for our defense's evaluation. The WeFDE technique allows us to determine how many bits of information each feature leaks for a given defense. The authors of the WeFDE paper made their code available to us, and for speed and memory requirements, we re-implemented most of the code ourselves and validated our common results against theirs. We perform information leakage analysis on undefended full-duplex traffic and defended traffic for 3043 features spread across 14 categories. The defenses we examine are WTF-PAD, W-T, and the full-duplex Case I variant of *Mockingbird*. The leakage for each feature and defense is shown in Figure 9. It is to note that our *Timing* and *Pkt. per Second* categories do not include W-T or *Mockingbird* measurements as the simulations for these defenses are unable to produce accurate timestamp estimations.

In general, we find that the information leakage of *Mockingbird* to be comparable to the other defenses. We find that any given feature leaks at most 1.9 bits of information in *Mockingbird.* W-T has similar maximum leakage, with at most 2.2 bits per feature, while WTF-PAD leaks at most 2.6 bits. The maximum amount of leakage seen for the undefended traffic was 3.4 bits, nearly twice that of *Mockingbird.* Our defense seems to be most vulnerable to the N-gram class of features, which is not surprising, as these features seem to be effective for all traffic defenses we examined. On the other hand, our defense is quite effective against packet count features, which was previously the most significant class of features for the undefended traffic and is also useful against W-T. Additionally, *Mockingbird* shows notable improvements over W-T in the *Transposition*, *First 30*, *Last 30*, and *Burst* features, while W-T is better than *Mockingbird* in the *Pkt. Distribution* feature.

Overall, the results of our information leakage analysis are largely in line with what we see in our Top-1 accuracy measurements: W-T and *Mockingbird* achieve similar performance, with *Mockingbird* slightly edging out W-T.

5.6 Results Analysis

We have presented a comparison of attack accuracies of *Mockingbird* against DF [36], AWF [34], CUMUL [27], k-FP [15], and k-NN [39] attacks for both Case I and Case II in Table 2. We also show the comparison of the performance of *Mockingbird* with two state-of-the-art defenses WTF-PAD [18] and Walkie-Talkie (W-T) [40]. In Table 3, we show the comparison of Top-2 accuracy of *Mockingbird* , WTF-PAD defense, and W-T defense.

Bandwidth Overhead. As described in Section 4.4, we designed *Mockingbird* 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, while the other proposed WF defenses have two-to-three times higher delays and bandwidth overheads.

From Table 2, we can see that, with full-duplex (FD) network traffic, the bandwidth overhead in Case I and Case II of Mockingbird are the same 56.5% and at least 7% and 15% lower than WTF-PAD and W-T, respectively. In contrast, the bandwidth overhead for half-duplex (HD) network traffic is lower only for Case I than WTF-PAD and W-T.

In HD network traffic, Case II requires at least 10% higher bandwidth overhead than Case I. In terms of bandwidth overhead, Case I is better for both FD and HD network traffic, although not to a significant degree. Overall, Case II for HD network traffic provides the strongest defense (28.8% accuracy) but requires the highest cost (73.5% bandwidth overhead) for any of the lightweight defenses.

In summary, our bandwidth overheads are better than WTF-PAD and W-T in most of the cases, except one. This 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 FD and HD network traffic are 35.2% and 35.5%, respectively, indicating that DF attack fails against our defense. In addition, the DF attack accuracies of *Mockingbird* are significantly lower than that of WTF-PAD, and slightly lower than that of W-T. More precisely, we can see from Table 2 that accuracies are at least 49% and 0.5% lower than that of WTF-PAD and W-T.

In Case II, the accuracy of the DF attack against FD network traffic is 55.5% which is higher than W-T but at least 34% lower than WTF-PAD. Surprisingly, the HD 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. In addition, the DF attack accuracy of Case II HD is at least 56% and at least 7% lower than that of WTF-PAD and W-T.

The AWF attack accuracies of *Mockingbird* for all the cases are lower than that of WTF-PAD and W-T defenses. The highest accuracy of AWF of *Mockingbird* is 34% which is still 5% lower than that of WTF-PAD and W-T defenses.

The CUMUL attack can achieve at best 36.8% accuracy against FD network traffic of Case I. In all the other cases, the CUMUL attack accuracies against our defense are closer to 30% or lower. We also want to point out that, CUMUL attack accuracies against our defense are lower than that of WTF-PAD and closer to W-T (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 *Mockingbird* is not limited to only against the deep-learning based DF attack, but also traditional machine-learning based CUMUL attack.

Though our defense has lower accuracy than WTF-PAD against k-FP attack, for two cases it is slightly higher than W-T, and and for one case equal to W-T. However, k-NN performs worst against our defense in all the cases than WTF-PAD and W-T.

It is also notable that both Case I and Case II provide consistencies between the attack accuracies for both FD and HD network traffic against all the attacks. It is interesting that all the attacks accuracies against Case I FD is lower than that of Case I HD. On the other hand, attacks accuracies against Case II FD is higher than that of Case II HD. Overall, the results demonstrate that *Mockingbird* is effective and has 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 Top-1 accuracy, which is normally referred to simply as the accuracy of the attack. Top-1 accuracy, however, does not provide a full picture of the effectiveness of a defense, as 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 *Mockingbird* against this threat, we compute the Top-2 accuracy and compare it to that of WTF-PAD and W-T. The results show that *Mockingbird* is somewhat 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 with 97% Top-2 accuracy as its defense design does not protect beyond confusion between two classes. In addition, WTF-PAD also has a very high 89.7% Top-2 accuracy. This Top-2 accuracy of *Mockingbird* indicates and ensures notably lower risk of de-anonymization of Tor client than WTF-PAD and W-T.

6 DISCUSSION

6.1 Deployment Challenges

Generating adversarial examples for WF defense is different than that of other domains such as computer vision or voice recognition. The main challenges we face in crafting adversarial examples for web traffic traces are as follows:

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 re-transmissions that lead to higher bandwidth and latency overhead, and harming the user's experience. To overcome this challenge, we introduce constraints in our algorithm to make the generated adversarial traces effective for WF defense purpose (see Section 4.4).

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 bridge in the network.

Crafting Adversarial Examples on the Fly. To craft the adversarial examples from traffic traces, we have to deal with the transmission of packets. Ideally, the traffic trace should be captured packet by packet and the adversarial examples should be generated in real time so that we can send and receive packets simultaneously. In addition, 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. However, we need a detector to be placed in both the client and the bridge (eventually in the Tor middle node) for *Mockingbird* to generate padding to fill out the adversarial traces. The detector is itself a deep-learning based model, meaning that the client and the bridge should have powerful computing capacity to generate traces fast enough to avoid any latency cost. Such a capability might not practical to assume, so we require an alternate way to make this defense work in the Tor network (see Section 6.2).

6.2 Alternate Deployment Plan

Here, we outline a method that may be practical for deploying the proposed defense.

- **Adversarial Traces on the Fly:** As the major constraint to deploy *Mockingbird* is training the detector, we think that it would be ideal if the Tor Project trains the detector and releases an updated trained detector in each of the version of Tor update. There are major two benefits for this approach: (i) the clients and bridges can continue their operation without requiring additional computing capacity, and (ii) over time, the detector will be trained on the traces of different sites and can incorporate any changes to website design or Tor traffic patterns.
- **Pre-generated Adversarial Traces:** If the computation requirements to produce adversarial traces on the fly are too much, patterns may instead be pre-generated. In this way, the adversarial traffic pattern of different websites would be made available to the client and the bridge with an update in each Tor version. Every time the detector is trained, we will get an updated traffic pattern for each site as well. The client and the bridge will maintain this pattern when sending any packet to each other. In particular, the client will know beforehand from the given traffic pattern how many packets in a burst it needs to send and how many of them are padding. On the other side, the bridge will also know how many fake packets to send to the client in a particular burst.

7 CONCLUSION

We propose a novel WF defense, *Mockingbird*, 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 lightweight defenses. *Mockingbird* uses a novel mechanism that adapts techniques to create adversarial traces against machine-learning and deep-learning classifiers. It generates adversarial traces that can significantly limit the ability of a WF adversary to distinguish a site, even though the adversary adopts adversarial training to train his classifier. Our defense mechanism results in 56% bandwidth overhead and drops the accuracy of the state-of-the-art WF attack from 95% to 29%-57%, depending on the scenario. 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 against the DF attack is at best 57%, whereas it is 90% for WTF-PAD and 97% for W-T. To gain a more objective perspective on the quality of our defense, we analyze the information leakage of website features. The results of this analysis are in-line with our previous conclusions when evaluating raw accuracy metrics. 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, 57% accuracy is very likely to lead to many false positives for the attacker. Overall, our findings indicate that *Mockingbird* can be a potential defense for Tor against WF attacks.

Availability

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

Acknowledgement

This material is based upon work supported by the National Science Foundation (NSF) under Grants Numbers 1423163, 1722743, and 1816851.

REFERENCES

[1] 2017. Alexa. (2017). http://www.alexa.com.

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

[3] Anish Athalye, Nicholas Carlini, and David Wagner. 2018. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International Conference on Machine Learning (ICML)*.

[4] Xiang Cai, Rishab Nithyanand, and Rob Johnson. 2014. CS-BuFLO: A Congestion Sensitive Website Fingerprinting Defense. In *Workshop on Privacy in the Electronic Society (WPES)*. ACM.

[5] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. 2014. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *ACM Conference on Computer and Communications Security (CCS)*.

[6] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. 2012. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *ACM Conference on Computer and Communications Security (CCS)*.

[7] Nicholas Carlini and David Wagner. 2017. Magnet and" efficient defenses against adversarial attacks" are not robust to adversarial examples. *arXiv preprint arXiv:1711.08478* (2017).

[8] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *Proceeding of the 38th IEEE Symposium on Security and Privacy (S&P)*.

[9] Nicholas Carlini and David A. Wagner. 2017. Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. In *AISec@CCS*.

[10] Heyning Cheng, , Heyning Cheng, and Ron Avnur. 1998. Traffic Analysis of SSL Encrypted Web Browsing. (1998).

[11] Giovanni Cherubin. 2017. Bayes, not Naïve: Security bounds on website fingerprinting defenses. *Privacy Enhancing Technologies Symposium (PETS)* (2017).

[12] Giovanni Cherubin, Jamie Hayes, and Marc Juarez. 2017. "Website fingerprinting defenses at the application layer". In *Privacy Enhancing Technologies Symposium (PETS)*.

[13] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. 2012. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *S&P*.

[14] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and Harnessing Adversarial Examples. (2014). arXiv:arXiv:1412.6572

[15] Jamie Hayes and George Danezis. 2016. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *USENIX Security Symposium*. USENIX Association.

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

[17] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. 2014. A Critical Evaluation of Website Fingerprinting Attacks. In *ACM Conference on Computer and Communications Security (CCS)*.

[18] Marc Juárez, Mohsen Imani, Mike Perry, Claudia Díaz, and Matthew Wright. 2016. Toward an Efficient Website Fingerprinting Defense. In *European Symposium on Research in Computer Security (ESORICS)*.

[19] Alex Kurakin, Dan Boneh, Florian Tramèr, Ian Goodfellow, Nicolas Papernot, and Patrick McDaniel. 2018. Ensemble Adversarial Training: Attacks and Defenses. In *International Conference on Learning Representations (ICLR)*.

[20] Shuai Li, Huajun Guo, and Nicholas Hopper. 2018. Measuring information leakage in website fingerprinting attacks and defenses. In *ACM Conference on Computer and Communications Security (CCS)*.

[21] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. 2016. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770* (2016).

[22] Xiapu Luo, Peng Zhou, EWW Chan, and Wenke Lee. 2011. HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. In *Network & Distributed System Security Symposium (NDSS)*.

[23] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR)*.

[24] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2015. DeepFool: a simple and accurate method to fool deep neural networks. (2015). arXiv:arXiv:1511.04599

[25] Rishab Nithyanand, Xiang Cai, and Rob Johnson. 2014. Glove: A Bespoke Website Fingerprinting Defense. In *Workshop on Privacy in the Electronic Society (WPES)*. ACM.

[26] Rishab Nithyanand, Xiang Cai, and Rob Johnson. 2014. Glove: A Bespoke Website Fingerprinting Defense. In *Workshop on Privacy in the Electronic Society (WPES)*. ACM.

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

[28] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *ACM Asia Conference on Computer and Communications Security*.

[29] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2015. The Limitations of Deep Learning in Adversarial Settings. (2015). arXiv:arXiv:1511.07528

[30] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2015. Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks. (2015). arXiv:arXiv:1511.04508

[31] Mike Perry. 2011. Experimental Defense for Website Traffic Fingerprinting. Tor Project Blog. (2011). https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting.

[32] Mike Perry. 2015. Padding Negotiation. Tor Protocol Specification Proposal. "https://gitweb.torproject.org/torspec.git/tree/proposals/254-padding-negotiation.txt". (2015). (accessed: February 15, 2019).

[33] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. 2018. Certified defenses against adversarial examples. In *International Conference on Learning Representations (ICLR)*.

[34] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. 2018. Automated Website Fingerprinting through Deep Learning. In *Network & Distributed System Security Symposium (NDSS)*.

[35] Vitaly Shmatikov and Ming-Hsiu Wang. 2006. Timing Analysis in Low-latency Mix Networks: Attacks and Defenses. In *European Symposium on Research in Computer Security (ESORICS)*.

[36] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. 2018. Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning. In *ACM Conference on Computer and Communications Security (CCS)*. ACM.

[37] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*.

[38] David Wagner and Bruce Schneier. 1996. Analysis of the SSL 3.0 Protocol. In *Proceedings of the 2nd Conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2 (WOEC'96)*. USENIX Association.

[39] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective Attacks and Provable Defenses for Website Fingerprinting. In *USENIX Security Symposium*. USENIX Association.

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

[41] CV V Wright, SE E Coull, and Fabian Monrose. 2009. Traffic morphing: An efficient defense against statistical traffic analysis. In *Network & Distributed System Security Symposium (NDSS)*.

A SCENARIOS OF EXISTING METHODS

In order to evaluate the efficacy of the adversarial traces generated by the Carlini and Wagner [8] 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 (Without-Adversarial-Training) or before the attack (With-Adversarial-Training). We evaluate both scenarios in the following sections. We use half-duplex (HD) closed-world dataset for these experiments. We explain the pre-processing of the data in Section 5.1.

A.1 SCENARIO I: Without-Adversarial-Training

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 [36].

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 4.

Table 4: Model Architecture of Simple CNN.

Layer type	size
Convolution + ReLU	$1 \times 8 \times 32$
Convolution + ReLU	$1 \times 8 \times 64$
Convolution + ReLU	$1 \times 8 \times 128$
Fully Connected + ReLU	512
Fully Connected + Softmax	number of classes

Table 5: The evaluation of the defenses against the state-of-the-art WF attacks as the attackers are trained on the undefended traces. BWO: Bandwidth Overhead, CNN is the simple CNN of Table 4.

	BWO	CNN	DF [36]	CUMUL [27]
Undefended	-	92%	98%	92%
Adversarial Traces	62%	2%	3%	31%

Table 5 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: With-Adversarial-Training

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 defends on the attacker's knowledge of the clients [17]. 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.

Table 6: The evaluation of the defenses against the state-of-the-art WF attacks as the attackers are trained on the defended traces. BWO: Bandwidth Overhead, CNN is the simple CNN of Table 4.

	BWO	CNN	DF [36]	CUMUL [27]
Undefended	-	92%	98%	92%
Adversarial Traces	62%	91%	97%	91%

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 4). Then we train the WF attacks, Simple CNN, DF, and CUMUL attacks, on 90% defended traces 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 6. 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, 97% and 91%, respectively. This means that generated adversarial traces are ineffective when the adversary adopts the technique of adversarial training. 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.