A Real-time Defense against Website Fingerprinting Attacks

Shawn Shan
University of Chicago

Arjun Nitin Bhagoji
University of Chicago

Haitao Zheng
University of Chicago

Ben Y. Zhao
University of Chicago

Abstract

Anonymity systems like Tor are vulnerable to Website Fingerprinting (WF) attacks, where a local passive eavesdropper infers the victim's activity. Current WF attacks based on deep learning classifiers have successfully overcome numerous proposed defenses. While recent defenses leveraging adversarial examples offer promise, these adversarial examples can only be computed after the network session has concluded, thus offer users little protection in practical settings.

We propose *Dolos*, a system that modifies user network traffic in real time to successfully evade WF attacks. *Dolos* injects dummy packets into traffic traces by computing *input-agnostic adversarial patches* that disrupt deep learning classifiers used in WF attacks. Patches are then applied to alter and protect user traffic in real time. Importantly, these patches are parameterized by a user-side secret, ensuring that attackers cannot use adversarial training to defeat *Dolos*. We experimentally demonstrate that *Dolos* provides 94+% protection against state-of-the-art WF attacks under a variety of settings. Against prior defenses, *Dolos* outperforms in terms of higher protection performance and lower information leakage and bandwidth overhead. Finally, we show that *Dolos* is robust against a variety of adaptive countermeasures to detect or disrupt the defense.

1 Introduction

Website-fingerprinting (WF) attacks are traffic analysis attacks that allow eavesdroppers to identify websites visited by a user, despite their use of privacy tools such as VPNs or the Tor anonymity system [30, 74]. The attacker identifies webpages in an encrypted connection by analyzing and recognizing network traffic patterns. These attacks have grown more powerful over time, improving in accuracy and scale. The most recent variants can overwhelm existing defenses, by training deep neural network (DNN) classifiers to identify the destination website given a network trace. In real world settings, WF attacks have proven effective in identifying traces in the wild from a large number of candidate websites using limited data [7, 66].

There is a long list of defenses that have been proposed and then later defeated by DNN based WF attacks. First, a class of defenses obfuscate traces by introducing randomness [23, 26, 36, 75]. These obfuscation-based defenses have been proven ineffective (< 60% protection) against DNN based attacks [7, 65]. Other defenses have proposed randomizing HTTP requests [17] or scattering traffic redirection across different Tor nodes [21, 31]. Again, these defenses provide poor protection against DNN-based attacks (<50% protection). Unsurprisingly, the success of DNN attacks have derailed efforts to deploy WF defenses on Tor (e.g., Tor stopped the implementation of WTF-PAD after it was broken by the DF attack [55, 65])

Against these strong DNN attacks, the only defenses to show promise are recent proposals that apply adversarial examples to mislead WF classification models [34, 58]. Adversarial examples are known weaknesses of DNNs, where a small, carefully tuned change to an input can dramatically alter the DNN's output. These vulnerabilities have been studied intensely in the adversarial ML community, and now are generally regarded a fundamental property of DNNs [35, 63].

Unfortunately, defenses based on adversarial examples have one glaring limitation. To be effective, adversarial examples must be crafted individually for each input [48]. In the WF context, the "input" is the entire network traffic trace. Thus a defense built using adversarial examples requires the entire traffic trace to be completed, before it can compute the precise perturbation necessary to mislead the attacker's WF classifier. This is problematic, since real world attackers will observe user traffic in real time, and be unaffected by a defense that can only take action after the fact.

In this paper, we propose *Dolos*, a practical and effective defense against WF attacks that can be applied to network traffic in real-time. The key insight in *Dolos* is the application of the concept of *trace-agnostic patches* to WF defenses, derived from the concept of adversarial patches. While adversarial examples are customized for each input, adversarial patches can cause misclassifications when applied to a wide range of input values. In the context of a WF defense,

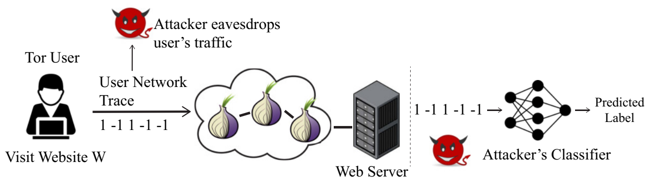

Figure 1: A WF attacker, positioned between user and the Tor network, eavesdrops on user network traffic. After the connection terminates, the attacker classifies the entire network trace using a pretrained WF attack classifier.

"patches" are pre-computed sequences of dummy packets that protect all network traces of visits to a specific website. A patch is applied to an active, ongoing network connection, i.e. in "real time." As we will show, *Dolos* generates patches parameterized by a user-side secret such as private keys. Unlike traditional patches or universal perturbations, patches parameterized by a user's secret cannot be overcome by attackers, unless they compromise the user's secret.

Our work describes experiences in designing and evaluating *Dolos*, and makes four key contributions:

- We propose *Dolos*, a new WF defense that is highly effective against the strongest attacks using deep learning classifiers. More importantly, *Dolos* precomputes patches before the network connection, and applies the patch to protect the user in real time.
- We introduce a secret-based parameterization mechanism for adversarial patches, which allows a user to generate randomized patches that cannot be correctly guessed without knowledge of the user-side secret. We show that this gives *Dolos* strong resistance to WF attacks that apply adversarial training with known adversarial patches and perturbations, which can otherwise defeat patches and universal perturbations [50].
- We evaluate *Dolos* under a variety of settings. Regardless of how the attacker trains its classifier (handcrafted features, deep learning, or adversarial training), *Dolos* provides 94+% protection against the attack. Compared to three state-of-the-art defenses [26, 36, 58], *Dolos* significantly outperforms all three in key metrics: overhead, protection performance, and information leakage.
- Finally, we consider attackers with full knowledge of our defense (i.e., full access to source code but not the user secret), and demonstrate that *Dolos* is robust against multiple adaptive attacks and countermeasures.

2 Web Fingerprinting Attacks and Defenses

2.1 Website Fingerprinting Attacks

In WF attacks, an attacker tries to identify the destination of user traffic routed through an encrypted connection to Tor or a VPN. The attacker passively eavesdrops on the user connection, and once the session is complete, feeds the network trace as input to a machine learning classifier to identify the website visited. Even though packets across the connection are encrypted and padded to the same size, attackers can distinguish traces as sequences of packets and their direction. The attacker's machine learning classifier is trained on packet traces generated by visiting a large, pre-determined set of websites beforehand.

Attacks via Hand-crafted Features. Panchenko *et al.* [53] proposed the first effective WF attack against Tor traffic using a support vector machine with hand-crafted features. Follow-up work proposed stronger attacks [11, 30, 32, 43, 44, 52, 70, 74] by improving the feature set and using different classifier architectures. The most effective attacks based on hand crafted features such as k-NN [74], CUMUL [52] , and k-FP [30] achieve over 90% accuracy in identifying websites based on network traces.

Attacks via DNN Classifiers. Recent work [1, 7, 62, 65] leverages deep neural networks (DNNs) to perform more powerful WF attacks. DNNs automatically extract features from raw network traces, and outperform previous WF attacks based on hand-crafted features. Two of the most successful DNN-based attacks are *Deep Fingerprinting (DF)* [65] and *VarCNN* [7]. *DF* leverages a deep convolutional neural network for classification, and reaches over 98% accuracy on undefended traces, or traces defended using existing defenses (§2.2). *Var-CNN* [7] further improves the attack performance by using a large residual network architecture, and can achieve high performance even with limited training data. Later in §6, our experiments show that *Dolos* is effective at defending against both of these state-of-the-art attacks, as well as against those using hand-crafted features.

2.2 Website Fingerprinting Defenses

WF defenses modify (add, delay, or reroute) packets to prevent the identification of the destination website using trace analysis. Broadly speaking, defenses either obfuscate traces based on expert-designed noise heuristics or leverage adversarial perturbations to evade machine learning classifiers.

Defenses via Trace Obfuscation. A number of defenses aim to obfuscate traces to increase the difficulty of classification. This obfuscation is performed either at the application layer or the network layer. Since these defenses do not specifically target the attacker's classifier, they generally afford low protection (<60%) against state-of-the-art WF attacks.

In *application layer obfuscation*, the defender introduces randomness into HTTP requests or the Tor routing algorithm (e.g., [18,21,31]). Application layer defenses generally make strong assumptions that are often unrealistic in practice, such as target websites implementing customized HTTP protocols [18] or only allowing attackers to observe traffic at a single Tor entry node [21, 31]. These defenses provide less than 60% protection against DNN-based attacks such as *DF* and *Var-CNN* [7, 65].

In *network layer obfuscation*, the defender inserts dummy packets into network traces to make website identification more challenging. First, early defenses [9, 10, 23] used constant-rate padding to reduce the information leakage caused by time gaps and traffic volume. However, these methods led to large bandwidth overhead (> 150%). Secondly, more recent work reduces the overhead by inserting packets in locations of the trace with large time gaps between packets (WTF-PAD [36]) or focusing on the front portion of the trace, which has been shown to contain the most information (FRONT [26]). However, WTF-PAD and FRONT only achieve 9% and 28% protection success respectively against strong WF attacks. Finally, supersequence defenses [51, 74, 75] try to find a *supersequence*, which is a longer packet trace that contains subsequences of different websites' traces. The strongest supersequence defense, Walkie-Talkie, achieves 50% protection against DNN attacks. Overall, against strong WF attacks, network layer obfuscation defenses either induce extremely large overheads (> 100%) or offer low protection (< 60%).

Defenses via Adversarial Perturbation. Goodfellow *et al.* first proposed evasion attacks against DNNs, where an attacker causes a DNN to misclassify inputs by adding small, *adversarial perturbations* [27] to them. Such attacks have been widely studied in many domains, e.g., computer vision [13, 15, 16, 39, 72], natural language processing [24, 83], and malware detection [28, 68]. Recent WF defenses [34, 58] use adversarial perturbations to defeat DNN-based attacks.

The challenge facing adversarial perturbation-based WF defense is that adversarial perturbations are computed with respect to a given input, which in the WF context is the full network trace. Thus, computing the adversarial perturbation necessary to protect a network connection requires the defender to know the *entire trace* before the connection is even made. This limitation renders the defense impractical to protect user traces in real-time.

Mockingbird [58] suggests that we can use a database of recent traces to compute perturbations, and use them on new traces. Yet it is widely accepted in ML literature that adversarial perturbations are input specific and rarely transfer [48]. We confirm this experimentally using Mockingbird's publicly released code: for the same website, perturbations calculated on one trace offer an average of 18% protection to a different trace from the same website. We tested 10 pairs of traces per website, over 10 websites randomly sampled from the same dataset used by Mockingbird [58].

In a concurrent manuscript, Nasr *et al.* [50] propose solving this limitation by precomputing *universal adversarial perturbations* for unseen traces, enabling the protection of live traces. However, an attacker aware of this defense can also compute these universal perturbations, and adversarially train their models against them to improve robustness. This countermeasure causes a significant drop in its protection to 76%. We show that our defense outperforms [50](§6.4).

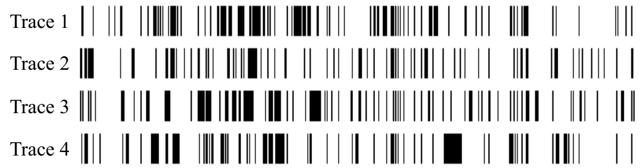

Figure 2: Example traces observed by a WF attacker when a Tor user u visits the same website. Each sample trace (x_u) records the packet direction (black bar: incoming packet, white bar: outgoing packet) of the first 500 packets in this session. We see that while the Tor user visits the same website in these four sessions, the traces observed by the attacker vary across sessions.

3 Problem and Threat Model

We consider the problem of defending against website fingerprinting attacks. A user u wishes to use the Tor network to visit websites privately. An attacker, after eavesdropping on u's traffic and collecting a trace of u's website visit (x_u), attempts to use x_u to determine (or classify) the destination website that u has just visited. To defend against such attacks, the defender seeks to inject "obfuscation" traffic into the Tor network, such that the traffic trace observed by the attacker ($\hat{x_u}$) will lead to a wrong classification result.

Threat Model. We use the same threat model adopted by existing WF attacks and defenses.

- The attacker, positioned between the user and Tor guard nodes, can only observe the user's network traffic but not modify it. Furthermore, the attacker can tap into user connections over a period of time and may see traffic on multiple website requests.
- We consider WF attacks that operate on packet directions[1]. This assumption is consistent with many previous defenses [21, 31, 34, 58, 75]. For each u's website session, the attacker collects its trace x_u as a sequence of packet directions (i.e., marking each outgoing packet as +1 and each incoming packet as -1). Figure 2 shows four examples when a Tor user visits the same website at different times. While the Tor user visits the same website in these four sessions, the traces observed by the attacker vary across sessions.
- We consider *closed-world* WF attacks where the user only visits a set of websites known to the attacker. These attacks are strictly stronger than *open-world* attacks where

[1]Since Tor's traffic is encrypted and padded to the same size [32], only packet directions and time gaps will leak information about the destination website.

the user can visit any website. The latter is far more challenging for the attacker.

- We assume the attacker has full knowledge of the defense method deployed by the user and/or Tor (i.e., access to source code). The attacker knows that the user traffic is protected by obfuscation, but does not know the run-time parameters (i.e., user-side secret) used to construct the obfuscation traffic.

Defender Capabilities. We make two assumptions about the defender.

- The defender can actively inject dummy packets into user traffic. With cooperation from both the user and a node in the Tor circuit (i.e., a Tor bridge[2]), the defender can inject packets in both directions (outgoing and incoming).
- The defender has no knowledge of the WF classifier used by the attacker.

Success Metrics. To successfully protect users against WF attacks, a WF defense needs to meet multiple criteria. First, it should successfully defend the most powerful known attacks (i.e., those based on DNNs), and reduce their attack success to near zero. Second, it should do so without adding unreasonable overhead to users' network traffic. Third, it needs to be effective in realistic deployments where users cannot predict the order of packets in a real-time network flow. Finally, a defense should be robust to adaptive attacks designed with full knowledge of the defense (i.e., with source code access).

Earlier in §2.2 we provide a detailed summary of existing WF defenses and their limitations with respect to the above success metrics.

4 A New Defense using Adversarial Patches

In this paper, we present *Dolos*, a new WF defense that exploits the inherent weaknesses of deep learning classifiers to provide a highly effective defense that protects network traces in real time, resists a variety of countermeasures, and incurs low overhead compared to existing defenses. In the following, we describe the key concepts behind *Dolos* and its design considerations. We present the detailed design of *Dolos* later in §5.

4.1 Background: Adversarial Patches

Our new WF defense is inspired by the novel concept of *adversarial patches* [8] in computer vision.

Adversarial patches are a special form of artifacts, which when added to the input of a deep learning classifier, cause an input to be misclassified. Adversarial patches differ from adversarial perturbations in that patches are both *input-agnostic* and *location-agnostic*, i.e., a patch causes misclassification

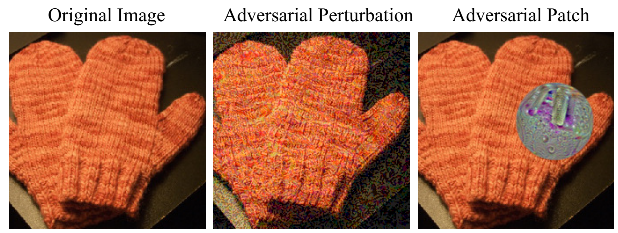

Figure 3: Sample images showing the difference between adversarial perturbations and adversarial patches.

when applied to an input, regardless of the value of the input or the location where the patch is applied [3]. Thus adversarial patches are "universal," and can be pre-computed without full knowledge of the input. Existing works have already developed adversarial patches to bypass facial recognition classifiers [67, 73, 80].

In computer vision, an adversarial patch is formed as a fixed size pattern on the image [8]. Figure 3 shows an example of an adversarial patch next to an example of adversarial perturbation. Given knowledge of the target classifier, one can search for adversarial patches under specific constraints such as patch color or intensity. For instance, for a DNN model ($\mathbb{F}$), an targeted adversarial patch p_{adv} is computed via the following optimization:

$$p_{adv} = \text{argmin}_p \underset{x \in X, l \in L}{\mathbb{E}} loss\left(\mathbb{F}(\Pi(p,x,l)), y_t\right) \qquad (1)$$

where X is the set of training images, L is a distribution of locations in the image, y_t is the target label, and Π is the function that applies the patch to a random location of an image. This optimization is performed over all training images in order to make the patch effective across images.

4.2 Adversarial Patches as a WF Defense

We propose to defend against WF attacks by adding *adversarial traffic patches* to user traffic traces, causing attackers to misclassify destination websites. To the best of our knowledge, our work is the first to use adversarial patches to defend against WF attacks.

Trace-agnostic. This is the key property of adversarial patches and why they are a natural defense against WF attacks. Given a user u and a website W, one can design a patch that works on any network trace produced when u visits W. We note that like adversarial perturbations, patches can be computed as the solution to a constrained optimization problem. Empirical studies have not found any limitations in the number of unique patches that can be computed for any targeted misclassification task.

Leveraging this property, the defender can pre-compute, for u, a set of W-specific adversarial patches. Once u starts to

[2] Tor bridges are often used to generate incoming packets for WF defenses [26, 36, 50, 58].

[3] We note that while input-dependent patches have been considered [37], they are largely utilized and analyzed in an input agnostic context.

visit the site W, the defender fetches a pre-computed patch and injects, in real-time, the corresponding dummy packets into u's live traffic. Furthermore, since patches are built using diverse training data, they are inherently robust against moderate levels of website updates and/or network dynamics. The defender periodically re-computes new patches or when they detect significant changes in website content and/or user networking environments.

4.3 Adversarial Patches for Network Traffic

Here, we describe new design considerations that arise from applying adversarial patches to network traffic traces.

For a specific user/website pair (user u, website W), the defender first uses some sample traces of u visiting W to compute a patch p for the pair. At run-time, when u initiates a connection to W, the defender fetches p and follows the corresponding insertion schedule to add dummy packets into u's Tor traffic. No original packets are dropped or modified.

Therefore, to generate adversarial patches for traffic traces, we follow the optimization process defined by Eq. (1), but change the Π function for patch injection. Note that the patches are generated for each specific user and website pair (u, W). Another key change is the "perturbation budget" (i.e., the maximum changes to the input), which is now defined as the bandwidth overhead introduced by those dummy packets. Patches for images are often limited by a small perturbation budget (e.g., $< 5\%$), because bigger patches are obvious to the human eye. Patches for traffic traces can support much larger budgets. For reference, Tor had deployed WTF-PAD as a WF defense, which incurs $50+\%$ bandwidth overhead [55, 65].

Strong Model Transferability of Patches. Here *model transferability* refers to the well-known phenomenon that ML classifiers trained for similar tasks share similar behaviors and vulnerabilities, even if they are trained on different architectures or training data [82]. Existing work has shown that when applied to an input, an adversarial perturbation or patch computed for a given DNN model will transfer across models [22, 56, 69]. In addition, a perturbed or patched input will also transfer to non-DNN models such as SVM and random forests [14, 22]. The level of transferability is particularly strong for large perturbation sizes [22, 56], as is the case for adversarial patches for network traffic.

Leveraging this strong model transferability, we can build a practical WF patch without knowing the actual classifier used by WF attackers. The defender can compute adversarial patches using local WF models, and they should succeed against attacks using other WF classifiers.

4.4 Proposed: Secret-based Patch Generation

A standard response by WF attackers to our patch-based defenses is to take patches generated by the defense, build and label patched user traces, and use these traces to (re)train an attack classifier to bypass the defense. In the adversarial ML literature, this is termed "adversarial training," and is regarded as the most reliable defense against adversarial attacks, including adversarial perturbations, adversarial patches, and universal perturbations.

In our experiments, we find that existing WF defenses fail under such attacks. For [21], protection drops to 55%. Noteworthy is the fact that [50], which uses universal perturbations, is particularly vulnerable to this attack. An attacker aware of the defense can compute the universal perturbation themselves, and use it for adversarial training. Our results show that the protection rate of [50] under adversarial training drops dramatically from 92% to 16%.

To resist adaptive attackers using adversarial training, we propose a novel secret-based patch generation mechanism that makes it nearly impossible for the attacker to reproduce the same patch as the defender. Specifically, the defender first computes a user-side secret $\mathcal{S}$ based on private keys, nonces, and website-specific identifiers, and then uses it to parameterize the optimization process of patch generation. The result is that different secrets will generate significantly different patches. When applied to the same network trace, the resulting patched traces will also display *significantly disjoint* representations in both input and feature spaces. Without access to the user-side secret, adversarial training using patches generated by the WF attacker will have little effect, and *Dolos* will continue to protect users from the WF attack.

5 *Dolos*

In this section we present the design of *Dolos*, starting from an overview, followed by detailed descriptions of its two key components: patch generation and patch injection.

5.1 Overview

Consider the task to protect a user u's visits to website W. *Dolos* implements this protection by injecting an adversarial patch $p_{W,T}$ to u's live traffic when visiting W, such that when the (defended) network trace is analyzed by a WF attacker, its classifier will conclude that u is visiting T (a website different from W). Here T is a configurable defense parameter.

Dolos includes two key steps: *patch generation* to compute an adversarial traffic patch ($p_{W,T}$) and *patch injection* to inject a pre-computed patch into u's live traffic as u is visiting W. This is also shown in Figure 4.

To generate a patch, *Dolos* inspects the WF feature space, and searches for potential adversarial patches that can effectively "move" the feature representation of a patched trace of u visiting W close to the feature representation of the (unpatched) traces of u visiting T. When these two feature representations are sufficiently similar, WF attacker classifiers

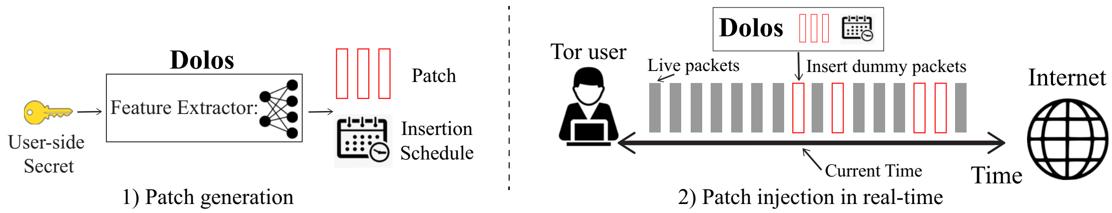

Figure 4: Our proposed *Dolos* system that protects a user u from WF attacks. *First*, *Dolos* precomputes a patch and its packet insertion schedule to protect u's visits to website W, using u's secret and a feature extractor. *Second*, when u is visiting website W, *Dolos* defends user's traces in real-time by inserting dummy packets according to the precomputed patch and schedule.

will identify the patched traces of u visiting W as a trace visiting T.

The above optimization can be formulated as follows:

$$p_{W,T} = \text{argmin}_p \mathbb{E}_{x \in X_W, x' \in X_T, s \in S} \mathbb{D}\left(\Phi(\Pi(p,x,s)), \Phi(x')\right) \quad \text{subject to } |p| \leq p_{budget} \tag{2}$$

where p_{budget} defines the maximum patch overhead, X_W (X_T) defines a collection of unpatched instances of u visiting W (T), S defines the set of feasible schedules to inject a patch into live traffic, and $\Pi(p,x,s)$ defines the *patch injection* function that injects a patch p onto the live traffic x under a schedule s. Finally, $\Phi(\cdot)$ refers to the local WF feature extractor used by *Dolos* while $\mathbb{D}$ is a feature distance measure (ℓ_2 norm in our implementation). Figure 5 provides an abstract illustration of the patch and the injection results.

Randomized Design. To prevent attackers from extracting or reverse-engineering our patches, we design *Dolos* to incorporate randomization into both patch generation and injection. In §5.2 and §5.3, we describe how *Dolos* uses user-side secret to configure T, S, $\Pi(p,x,s)$ and the optimization process of Eq. (2) to implement patches that are robust against adaptive attacks.

Choosing $\Phi(\cdot)$. As discussed earlier, *Dolos* does not assume knowledge of the WF classifiers used by the attackers. Instead, *Dolos* operates directly on the *feature space* and uses a feature extractor Φ, basically a partial neural network trained on the same or similar task. *Dolos* can train Φ locally or use a pre-trained WF classifier from a trusted party (e.g., Tor). Given an input x, *Dolos* uses the outputs of an intermediate layer of Φ as x's feature vector that quantifies distance in the feature space. A well-trained Φ can help *Dolos* tolerate web content dynamics and can function well with a wider range of websites both known and unknown [62].

5.2 Patch Generation

The patch generation follows the optimization process defined by eq. (2). Here a novel contribution of *Dolos* is to use a secret to configure the patch generation process, so that the resulting patches are strictly conditioned on this secret. The means that when we apply patches generated with different secrets to the same original trace, the resulting patched traces will be significantly different in both input and WF feature spaces. The secret can be a one way hash of a user's private key, a time-based nonce, and an optional website specific modifier. This secret allows *Dolos* to compute multiple, distinct patches based on specific users, destination websites, and time. A defender can periodically recompute patches with updated nonces to prevent any longitudinal attacks that try to identify a common patch across multiple connections to the same website. This controlled randomization prevents attackers from observing the true distribution of the patched traces over time or multiple traces.

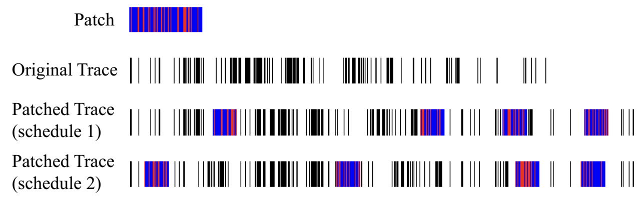

Figure 5: An abstract illustration of adversarial traffic patch and how it is injected into user traffic traces. Black/white bars mark the packet directions (out/in) of original packets, red/blue mark the (out/in) directions of dummy packets (i.e., the patch). *Dolos* injects the patch into a live user trace following a specific schedule. Here we show the results when the same patch is injected using two different schedules (1 & 2) and random packet flipping (see §5.3).

Parameterized Patch Generation. When generating a patch, *Dolos* uses a secret $\mathcal{S}$ to determine T (the target website) and the exact number of dummy packets to be injected, i.e., the length of the patch $|p_{W,T}|$.

Choosing T: *Dolos* collects a large pool of candidate target websites (400,000 in our implementation) from the Internet. To protect user u's visit to W, *Dolos* uses u's secret $\mathcal{S}$ to "randomly" select a website from the candidate pool, whose feature representation is far from that of the original trace, i.e., $\mathbb{E}_{x \in X_W} \Phi(x)$ is largely dissimilar from $\mathbb{E}_{x \in X_T} \Phi(x)$. For our implementation, we first calculate the feature distance (ℓ_2 norm) between W and each candidate in our large pool, identify the top 75$^{\text{th}}$ percentile as a reduced candidate pool for W, and use the secret $\mathcal{S}$ as a random seed to select one from the reduced pool.

Choosing $|p_{W,T}|$: To further obfuscate the appearance of our patches, we use the same secret $\mathcal{S}$ to select the patch length (i.e., the number of dummy packets to be injected into the user traffic). Specifically, given p_{budget} (the maximum overhead ratio), we "randomly" pick a patch length between $(p_{budget} - \varepsilon, p_{budget})$ where ε is a defense parameter (0.1 in our implementation).

Patch Optimization. When solving the patch optimization problem defined by eq. (2), we use Stochastic Gradient Descent (SGD) [64] by batching samples from X_W. We note that in order to apply SGD, we need to relax the constraints on $p_{W,T}$ and allow it to lie in $[-1,1]^n$, i.e., we allow for continuous values between -1 and $+1$ while optimizing. When the optimization ends, we quantize any negative value as -1 and any non-negative value as $+1$. This method is widely used to solve discrete optimization problems in domains like natural language processing [54, 61, 76] and malware classification [38], which avoids the intractability of combinatorial optimization [40]. Finally, we note that the optimization also takes into account the patch injection process (e.g., $\Pi(p,x,s)$), which we describe next in §5.3.

Managing Secrets. Note that *Dolos* manages secrets and their original components (private keys, nonces, website specific identifiers) following the standard private key management recommendations [5]. Secrets are only stored on user's device and updated periodically.

5.3 Patch Injection

The patch injection component has two goals: 1) making the patch input-agnostic and location-agnostic, in order to deploy our WF defense on live traffic, and 2) obfuscating the patched traces at run-time to prevent attackers from detecting/removing patches from the observed traces (to recover the original trace), or dismantling patches via trace segmentation to reduce its coverage/effectiveness.

Existing solutions (used by adversarial patches for images) do not meet these goals. They simply inject a given patch (as a fixed block) at any location of the trace. Under our problem context, an attacker can easily recognize the fixed pattern by observing the user traces over a period of time. Instead, we propose to combine a segment-based patch injection method with run-time packet-level obfuscation.

Segment-based Patch Injection. A pre-computed patch p is a sequence of dummy packets (+1s and -1s) designed to be injected into the user's live traffic x. We first break p into multiple segments of equal size M_p (e.g., 30 packets), referred to as "mini-patches." Each mini-patch is assigned to protect a segment of x of size M_x (e.g., 100 packets). We configure the patch generation process to ensure that, within each segment of x, the corresponding mini-patch stays as a fixed block. Patches are *location-agnostic*, so they will produce the same effect regardless of location within the segment. Therefore, given M_x and M_p, the injection function $\Pi(p,x,s)$ in Eq. (2) depends on s, an injection schedule that defines the randomly chosen location of each mini-patch within x's segments (see Figure 5). S defines all possible sets of mini-patch locations.

An advantage of splitting up the patch into mini-patches is that it protects against attackers trying to infer the website by searching for subsequences of packets. Our results confirm this hypothesis later in §7.1.

Run-time Patch Obfuscation using Packet Flips. When a single patch p is used to protect u's visit to W over some window of time, the same p could appear in multiple patched traces. While our segment-based injection hides p within the patched traces, it does not change p. Thus a resourceful attacker could potentially recover p using advanced trace analysis techniques. To further elevate the obfuscation, we apply random "packet flipping" to make p different in each visit to W. Specifically, in each visit session, we randomly choose a small set of dummy packets (in p) and flip their directions (out to in, in to out). We ensure that this random flipping operation is accounted by the patch generation process (eq. (2)), so that it does not affect the patch effectiveness. Later in §7, we show that this random flipping does deter countermeasures that leverage frequency analysis to estimate p.

Deployment Considerations. At run-time, the user and Tor bridge follow a simple protocol to protect traces. As soon as the user u requests a website W, *Dolos* sends the precomputed patch $p_{W,T}$ (after random flipping) and the current insertion schedule s to the Tor bridge through an encrypted and fixed length (padded) network tunnel. Next, the user and the Tor bridge coordinate to send dummy packets to each other to achieve the protection.

6 Evaluation

In this section, we perform a systematic evaluation of *Dolos* under a variety of WF attack scenarios. Specifically, we evaluate *Dolos* against i) the state-of-the-art DNN WF attacks whose classifiers use either the same or different feature extractors of *Dolos* (§6.2), ii) non-DNN WF attacks that use handcrafted features (§6.3). Furthermore, we compare *Dolos* against existing WF defenses under these attacks (§6.4) and in terms of *information leakage*, which estimates the number of potential vulnerabilities facing any WF defense (including those not yet exploited by existing WF attacks) (§6.5).

Overall, our results show that *Dolos* is highly effective against state-of-the-art WF attacks ($\geq 94.4\%$ attack protection at a 30% bandwidth overhead). It largely outperforms existing defenses in all three key metrics: protection success rate, bandwidth overhead, and information leakage.

Finally, under a simplified 2-class setting, we show that *Dolos* is *provably robust* with a sufficiently high bandwidth overhead. Our theory result and proof, which rely on the theory of optimal transport, are listed in the Appendix.

6.1 Experimental Setup

WF Datasets. Our experiments use two well-known WF datasets: `Sirinam` and `Rimmer` (see Table 1), which are commonly used by prior works for evaluating WF attacks and defenses [7,33,50,62,65]. Both datasets contain Tor users' website traces (as data) and their corresponding websites (as labels). `Sirinam` was collected by Sirinam *et al.* around February 2016 and includes 86,000 traces belonging to 95 websites [65] in the Alexa top website list [3]. `Rimmer` was collected by Rimmer *et al.* [62] in January 2017, covering 2 million traces for visiting Alexa's top 900 websites. The two datasets have partial overlap in their labels. Following prior works on WF attacks and defenses, we pad each trace in the datasets into a fixed length of 5000.

Dataset Name	# of Labels	# of Training Traces	# of Testing Traces
`Sirinam`	95	76K	10K
`Rimmer`	900	2M	257K

Table 1: Two WF datasets used by our experiments.

***Dolos* Configuration.** Next we describe how we configure *Dolos*' feature extractor Φ, the patch injection parameters, and the user-side secret. We build four feature extractors using two well-known DNN architectures for web trace analysis, Deep CNN (from the DF attack) and ResNet-18 (from the Var-CNN attack), and train them on the above two WF datasets. We also apply standard adversarial patch training [4, 50, 60] to fine-tune these feature extractors for 20 epochs, which helps increase the transferability of our adversarial patches[4]. Table 2 lists the resulting Φs and we name them based on the model architecture and training dataset.

Model Architecture $\mathbb{F}$	Training Data $\mathcal{X}$	Feature Extractor Φ
Deep CNN (DF)	`Sirinam`	$\mathtt{DF}_{\mathtt{Sirinam}}$
ResNet-18 (Var-CNN)	`Sirinam`	$\mathtt{VarCNN}_{\mathtt{Sirinam}}$
Deep CNN (DF)	`Rimmer`	$\mathtt{DF}_{\mathtt{Rimmer}}$
ResNet-18 (Var-CNN)	`Rimmer`	$\mathtt{VarCNN}_{\mathtt{Rimmer}}$

Table 2: The four feature extractors (Φ) used in our experiments, their model architecture and training dataset.

When injecting patches, we set the mini-patch length M_p to 10 and the trace segment length M_x to M_p/R, where R represents the bandwidth overhead of the defense ($0 < R < 1$). We experiment with different values of M_p and find that its impact on the defense performance is insignificant. Thus we empirically choose a value of 10. By default, we set the packet flipping rate $\beta = 0.2$, i.e., at run-time 20% of the dummy packets in each mini-patch will be flipped to a different direction. The impact of β on possible countermeasures is discussed later in §7.1. We use a separate dataset that contains traces from 400,000 different websites as our pool of target websites [62].

Finally, since our patch generation depends on the user-side secret, we repeat each experiment 10 times, each using a randomly formulated user-side secret (per website). We report the average and standard deviation values. Overall, the standard deviations are consistently low across our experiments, i.e., $< 1\%$ for protection success rate.

Attack Configuration. We consider two types of WF attacks: 1) non-DNN based and using handcrafted features, i.e., *k-NN* [74], *k-FP* [30], and *CUMUL* [52], and 2) DNN-based attacks, i.e., *DF* [65] and *Var-CNN* [7]. We follow the original implementations to implement these attacks.

Consistent with prior WF defenses [21, 26, 33, 58, 75], we assume that the attacker trains their classifiers using *defended traces*, i.e., those patched using our defense. To generate and label such traces, the attacker downloads *Dolos* and runs it on their own traces when visiting a variety of websites. Here the attacker must input some user-side secret to run *Dolos*, which we refer to as $\mathcal{S}_{attack}$. Since the attacker has no knowledge of the user-side secret $\mathcal{S}_{defense}$ being used by *Dolos* to protect the current user u, we have $\mathcal{S}_{defense} \neq \mathcal{S}_{attack}$. A relevant countermeasure by attackers is to enumerate many secrets when training the attack classifier, which we discuss later in §7.3.

Intersection Attacks. We also consider *intersection attacks* in our evaluation of *Dolos*. Here the attacker assumes the victim visits the same websites regularly and monitors the victim's traffic for a longer time period (e.g., days). With these information, the attacker could make better inferences on user's traces. We test *Dolos* against the intersection attack used by a previous WF defense [58] and find that the attack is ineffective on *Dolos*. More details about our experiments and results can be found in the Appendix.

Evaluation Metrics. We test *Dolos* against various WF attacks using the testing traces in the two WF datasets (see Table 1). We evaluate *Dolos* using three metrics: 1) *protection success rate* defined as the WF attack's misclassification rate on the defended traces, 2) *bandwidth overhead* $R = \frac{\text{patch length}}{\text{original trace length}}$, and 3) *information leakage*, which measures the amount of potential vulnerability of any WF defense [17, 42].

We also examine the computation cost of *Dolos*. The average time required to compute a patch is 19s on an Nvidia Titan X GPU and 43s on an eight core i9 CPU machine.

6.2 *Dolos* vs. DNN-based WF Attacks

Our experiments consider three attack scenarios.

- *Matching attack/defense*: the attacker and *Dolos* operate on the same original trace data $\mathcal{X}$ (e.g., `Sirinam`) and use the same model architecture $\mathbb{F}$ (e.g., DF). *Dolos* trains its feature extractor Φ using model $\mathbb{F}$ and data $\mathcal{X}$, while the

[4] We show the protection results of *Dolos* when using standard feature extractors without adversarial patch training in table 9 in the Appendix.

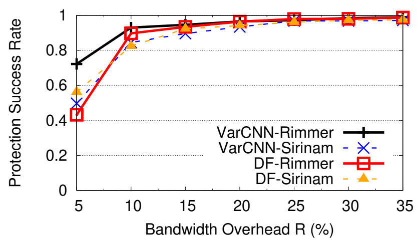

Figure 6: When against DNN-based attacks, *Dolos*'s protection success rate increases rapidly with its bandwidth overhead R. *Dolos* achieves $> 97\%$ protection rate when R reaches 30%. Assuming matching attack/defense.

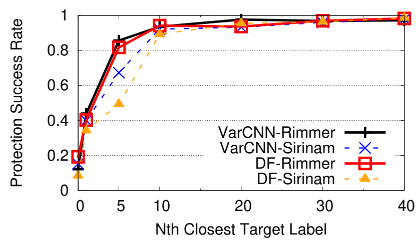

Figure 7: Worst-case analysis on the impact of key collision on *Dolos*, where the target feature representations T_{attack} and $T_{defense}$ are N^{th} nearest neighbors in the feature space. Assuming matching attack/defense.

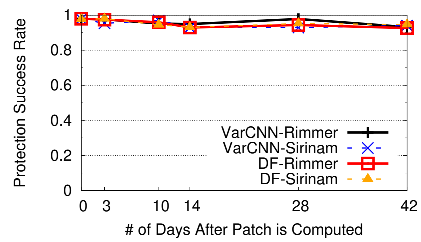

Figure 8: *Dolos*'s effectiveness over time against fresh attacks after deploying a patch to protect u visiting W at day 0. The same patch is able to resist attacks that train their classifiers on newly generated defended traces in subsequent days.

Dataset	*Dolos*'s Feature Extractor	*Dolos*'s Protection Success Rate (%) against WF Attacks				
		k-NN	k-FP	CUMUL	DF	Var-CNN
Sirinam	$\texttt{DF}_{\texttt{Sirinam}}$	98.2 ± 0.2	98.7 ± 0.5	97.3 ± 0.4	$\mathbf{97.2 \pm 0.7}$	96.4 ± 0.9
	$\texttt{DF}_{\texttt{Rimmer}}$	97.4 ± 0.5	98.2 ± 0.6	96.3 ± 0.6	95.1 ± 0.8	97.2 ± 0.5
	$\texttt{VarCNN}_{\texttt{Sirinam}}$	97.7 ± 0.4	98.0 ± 0.6	95.8 ± 0.8	94.4 ± 0.6	$\mathbf{96.8 \pm 0.4}$
	$\texttt{VarCNN}_{\texttt{Rimmer}}$	97.5 ± 0.6	98.9 ± 0.3	97.4 ± 0.2	95.9 ± 0.8	95.4 ± 0.4
Rimmer	$\texttt{DF}_{\texttt{Sirinam}}$	98.8 ± 0.3	97.6 ± 0.5	96.5 ± 0.3	96.3 ± 0.5	97.3 ± 0.6
	$\texttt{DF}_{\texttt{Rimmer}}$	97.0 ± 0.5	97.6 ± 0.6	97.8 ± 0.6	$\mathbf{98.0 \pm 0.4}$	97.9 ± 0.6
	$\texttt{VarCNN}_{\texttt{Sirinam}}$	98.2 ± 0.4	98.9 ± 0.2	97.3 ± 0.5	95.9 ± 0.6	97.3 ± 0.4
	$\texttt{VarCNN}_{\texttt{Rimmer}}$	98.6 ± 0.3	98.1 ± 0.4	98.4 ± 0.3	96.8 ± 0.5	$\mathbf{98.4 \pm 0.5}$

Table 3: *Dolos*'s protection success rate against different WF attacks. The **bold** entries are results under matching attack/defense.

attacker trains its attack classifier using model $\mathbb{F}$ and the *defended* version of $\mathcal{X}$.

- *Mismatching attack/defense*: the attacker and *Dolos* use different $\mathcal{X}$ and/or $\mathbb{F}$ when building their attack classifier and feature extractor, respectively.
- *Defense effectiveness over time*: *Dolos* starts to apply a patch p to protect u's visits to W since day 0; the attacker runs *fresh* WF attacks in the subsequent days by training attack classifiers on defended traces freshly generated in each day.

Scenario 1: Matching Attack/Defense. Figure 6 plots *Dolos*'s protection success rate against its bandwidth overhead, for each of the four $(\mathbb{F}, \mathcal{X})$ combinations listed in Table 2. The standard deviation values are small ($< 1\%$) and thus omitted from the figure for clarity.

We see that *Dolos* consistently achieves 97% or higher protection rate when the defense overhead $R \geq 30\%$. This result confirms that the adversarial patches, when controlled via user-side secrets, can effectively break WF attacks trained on defended traces. Note that *Dolos* is even more effective against WF attacks whose classifiers are trained on original (undefended) traces, i.e., 98% protection success rate with a 15% overhead. For the rest of the paper, we use $R = 30\%$ as the default configuration.

Likelihood of Secret Collisions: In the above experiments, we randomly select the secret pair $(\mathcal{S}_{attack}, \mathcal{S}_{defense})$ and show that in general, as long as $\mathcal{S}_{attack} \neq \mathcal{S}_{defense}$, *Dolos* is highly effective against WF attacks. Next, we also run a *worst-case* analysis that looks at cases where the combination of $\mathcal{S}_{attack}$ and $\mathcal{S}_{defense}$ leads to heavy collision in the WF feature space. That is, the patches generated by $\mathcal{S}_{attack}$ and $\mathcal{S}_{defense}$ will move the feature representation of the original trace to the target feature representations of website T_{attack} and $T_{defense}$, respectively, but the two targets are close in the feature space (with respect to the ℓ_2 distance). Here we ask the question: *how "close" do T_{attack} and $T_{defense}$ need to be in order to break our defense*?

We answer this question in Figure 7 by plotting *Dolos*'s protection success rate when T_{attack} is the N^{th} nearest label to $T_{defense}$ in the feature space ($N \in [1, 40]$). Here we show the result for each of the four $(\mathbb{F}, \mathcal{X})$ combinations. We see that as long as T_{attack} is beyond the top 20^{th} nearest neighbors of $T_{defense}$, *Dolos* can maintain $> 96\%$ protection success rate. Since our pool of target websites is very large (400,000 websites), the probability of an attacker finding a K_{attack} that weakens our defense is very low ($p_{\text{bad}} \approx \frac{20}{400{,}000} = 5 \times 10^{-5}$). Later in §7, we show that even when the attacker trains their classifiers on defended traces produced by a large number of secrets, the impact on *Dolos* is still minimum.

Scenario 2: Mismatching Attack/Defense. We now con-

Dataset	Defense Name	Bandwidth Overhead	Protection Success Rate Against WF Attacks					
			k-NN	k-FP	CUMUL	DF	Var-CNN	Worst Case
`Sirinam`	WTF-PAD	54%	87%	56%	69%	10%	11%	10%
	FRONT	80%	96%	68%	72%	31%	34%	31%
	Mockingbird	52%	94%	89%	91%	69%	73%	69%
	Dolos	**30%**	**98%**	**99%**	**97%**	**96%**	**95%**	**95%**
`Rimmer`	WTF-PAD	61%	84%	58%	72%	14%	11%	11%
	FRONT	72%	97%	62%	68%	37%	39%	37%
	Mockingbird	57%	96%	87%	90%	71%	79%	71%
	Blind Adversary	11%	-	-	-	-	76%*	76%
	Dolos	**10%**	**95%**	**92%**	**93%**	**87%**	**92%**	**87%**
	Dolos	**30%**	**99%**	**98%**	**98%**	**97%**	**98%**	**97%**

Table 4: Comparing bandwidth overhead and protection success rate of WTF-PAD, FRONT, Mockingbird, Blind Adversary, and *Dolos*. * we take the number from their original paper [50] as the authors have not released their source code.

sider the more general scenario where the attacker and *Dolos* use different $\mathcal{X}$ and/or $\mathbb{F}$ to train their classifiers and feature extractors, respectively. Here we consider two existing DNN-based WF attacks, DF and Var-CNN, trained on `Sirinam` or `Rimmer`, and configure *Dolos* to use one of the four feature extractors listed in Table 2. Using the test data of `Sirinam` and `Rimmer`, we evaluate *Dolos* against these WF attacks (DF and Var-CNN), and list *Dolos*'s protection success rate in Table 3. As reference, we also include the results of matching attack/defense (scenario 1), marked in bold.

Overall, *Dolos* remains highly effective ($> 94\%$ protection rate) against attacks using different training data and/or model architecture. This shows that our proposed adversarial patches generated from a local feature extractor can successfully *transfer* to a variety of attack classifiers.

Scenario 3: Defense Effectiveness Over Time. Next, we evaluate *Dolos* against freshly generated attacks over time. Here *Dolos* computes and deploys a patch p to protect u's visits to W at day 0; the attacker continues to run WF attacks in the subsequent days, and each day, they train the attack classifier using defended traces generated on the current day. We use this experiment to examine the robustness of *Dolos*'s patches under web content dynamics and network dynamics, also referred to as concept drift.

Our experiment uses the concept drift dataset provided by Rimmer *et al.* [62], which was collected along with `Rimmer`. This new dataset consists of 200 websites (a subset of 900 websites in `Rimmer`), repeatedly collected over a six week period (0 day, 3 days, 10 days, 14 days, 28 days, 42 days). We run *Dolos* using each of the four feature extractors to produce a patch at day 0. The attacker uses the Var-CNN classifier and trains the classifier using fresh defended traces generated on day 3, 10, 14, 28 and 42. We see that the protection success rate remains consistent ($> 93\%$) over 42 days.

6.3 *Dolos* vs. non-DNN Attacks

While *Dolos* targets the inherent vulnerability of DNN-based WF attacks, we show that the adversarial patches produced by *Dolos* are also highly effective against non-DNN based WF attacks. Here we consider three most-effective non-DNN attacks: k-NN [74], k-FP [30], and CUMUL [52]. Table 2 lists the protection success rate of *Dolos* under these attacks, using each of the four different feature extractors. Our results align with existing results: adversarial patches designed for DNN models also transfer to non-DNN models [14, 22].

6.4 Comparison with Previous WF Defenses

Table 4 lists the performance of *Dolos* and four state-of-the-art defenses (WTF-PAD [36], FRONT [26], Mockingbird [58], Blind Adversary [50] as described in §2). We evaluate them against five attacks on the two WF datasets. For *Dolos*, we use $\texttt{VarCNN}_{\texttt{Rimmer}}$ as the local feature extractor.

WTF-PAD. WTF-PAD is reasonably effective against traditional ML attacks, but performs poorly against any DNN based attack, i.e., protection success rate drops to 10%. These findings align with existing observations [36, 65].

FRONT. FRONT is effective against non-DNN attacks but fails against DNN attacks. In our experiments, FRONT induces larger bandwidth overhead than reported in the original paper [26]. The discrepancy is because FRONT's overhead is dataset dependent and a separate paper [33] reports the same overhead as ours when applying FRONT to `Sirinam`.

Mockingbird. Mockingbird is effective against non-DNN attacks and reasonably effective against DNN attacks (71% protection success). As stated earlier, the biggest drawback of Mockingbird is that the defense needs the access to the full trace beforehand, making it unrealistic to implement in the real-world.

Blind Adversary. We are not able to obtain the source code of Blind adversary at the time of writing. In the original paper, Blind Adversary is evaluated on the same dataset (`Rimmer`) using same robust training technique. However, the authors report the results for 11% overhead under the *white-box* setting. Using the same setting and similar overhead, *Dolos* achieves 92% protection success rate whereas Blind Adversary achieves 76% protection.

6.5 Information Leakage Analysis

Recent works [17,42] argue that WF defenses need to be evaluated beyond protection success rate because existing WF attacks may not show the hidden vulnerability of the proposed defense. Li *et al.* [42] proposes an information leakage estimation framework (WeFDE) that measures a given defense's information leakage on a set of features. We measure *Dolos* information leakage following the WeFDE framework.

WeFDE computes the mutual information of trace features between different classes. A feature has a high information leakage if it distinguishes traces between different websites. However, we cannot directly apply such analyzer to measure the leakage of *Dolos* because *Dolos* does not seek to make traces from different websites indistinguishable from each other. Attacker can separate the traces defended by *Dolos* but has no way of knowing which websites the traces belong to.

Thus to measure the information leakage of *Dolos*, we need to obtain the overall distribution of defended traces agnostic to secrets, i.e., defended traces using all possible secrets. We approximate this distribution using traces generated with a large number of secrets and feed the aggregated traces to WeFDE for information leakage analysis.

We use all websites from `Rimmer`. For each website, we use $\texttt{DF}_{\texttt{Sirinam}}$ and 80 different secrets to generate defended traces. We find that enumerating more than 80 secrets has limited impact on the information leakage results. We aggregate the traces together before feeding into WeFDE. We compare *Dolos* with three previous state-of-the-art defenses, WTF-PAD, FRONT, and MockingBird. We measure the leakage (bits) on two sets of features: 1) hand crafted features from the WeDFE paper [42], and 2) DNN features from a model trained on the defended traces of a given defense.

Leakage on Hand Crafted Features. We first measure the information leakage on the default set of 3043 hand crafted features from WeFDE. This feature set covers various categories, e.g., packet ngrams, counts, and bursts. Figure 9 shows the empirical cumulative distribution function (ECDF) of information leakage across the features. The curve of *Dolos* increases much faster than the curves of other defenses. For *Dolos*, no feature leaks more than 1.7 bits of information, which is lower than the minimum leakage of other defenses.

Leakage on Features Trained on Defended Traces. For each defense, we measure the information leakage on the feature space of Var-CNN models trained on the defended traces. For *Dolos*, the defended traces used by the attacker are generated by a different secret. Figure 10 shows the ECDF of the leakage across all the features. Overall, this feature set leaks more information than the hand crafted features in Figure 9. Again the curve of *Dolos* increases faster than the curves of other defenses. The gap between *Dolos* and other defenses is larger in this set of features showing *Dolos* is more resilient against models trained on defended traces due to randomness introduced by the secret.

7 Countermeasures

In this section, we explore additional countermeasures that could be launched by attackers with complete knowledge of *Dolos*. We consider three classes of countermeasures: detecting *Dolos* patches, preprocessing inputs to disrupt *Dolos* patches, and boosting WF classifier robustness. Unless otherwise specified, experiments in this section run VarCNN-based attacks on the `Rimmer` dataset, and the defender uses the $\texttt{DF}_{\texttt{Sirinam}}$ feature extractor to generate patches (see §6).

7.1 Detecting *Dolos* Patches

An attacker can apply data analysis techniques to detect the presence of patches in network traces. Detection can lead to possible identification of patches and their removal.

Frequency Analysis. An attacker who observes multiple visits to the same website by the same user might identify defender's patch sequences using frequency analysis, if the same patch is applied to multiple traces over a period of time. In practice, this frequency analysis might be challenging because: the location of the patch is randomized, *Dolos* randomly flips a subset of the dummy packets each time the patch is applied, and packet sequences in patches might blend in naturally with unaltered network traces.

We test the feasibility of this countermeasure. We assume the attacker has gathered network traces from 100 separate visits by the same user to a single website. For each trace, the attacker enumerates all packet sequences with the length of the mini-patch (known to attacker). To address the random flipping, the attacker merges packet sequences that have a Hamming distance smaller than the flipping ratio (known to the attacker). This produces a set of packet sequences for each trace. The sequence of each mini-patch should appear in every set.

As the flip ratio increases, however, packet sequences from the patch start to blend in with common packet sequences found frequently in benign (unpatched) network traces. Patch sequences can thus blend in with normal sequences, making their identification and removal difficult.

For example, for each website in `Rimmer`, we take 100 original network traces and perform frequency analysis. With a flip ratio of 0.2, an website has on average 45% of its packet sequences showing up in every set as false positives that look like potential patches. An aggressive attacker can remove *all* such high frequency packet sequences before classifier inference. Removing these high frequency packet sequences means the attacker's classifier accuracy is reduced down to 7%. We perform this test using different values of flip ratio β and show the results in Table 8. success rate against a normal

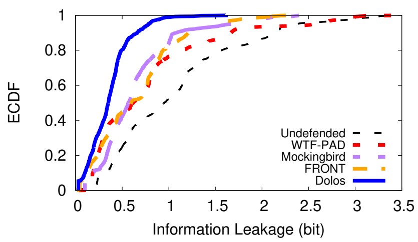

Figure 9: The ECDF of information leakage on hand crafted features from WeFDE.

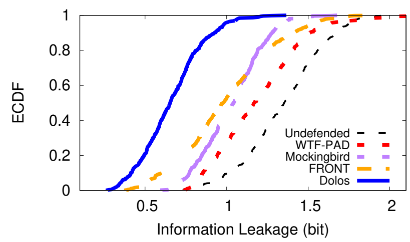

Figure 10: The ECDF of information leakage on features from models trained on defended traces.

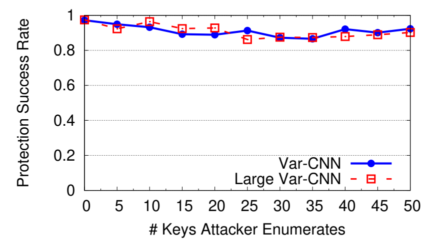

Figure 11: Protection performance drops slightly as attacker trains on traces defended by increasing numbers of secrets.

Ratio of Packets	Protection Success Rate When	
	Drop Packets	Flip Packets
0.05	97%	97%
0.1	96%	97%
0.2	95%	96%
0.3	96%	96%
0.5	96%	96%

Table 5: Protection success rate remains high as attacker drops or flips a portion of packets before classification.

Ratio of Packets Trimmed	Protection Success Rate
0.1	97%
0.2	97%
0.3	96%
0.5	96%

Table 6: Protection success rate remains high as attacker trim a portion of packets at the end of each trace before classification.

# of Epochs Trained with Adversarial Training	Protection Success Rate
20	96%
40	95%
100	94%
200	95%

Table 7: Protection success rate remains high when transferring to increasing more robust models.

Flip Ratio	Protection Success Var-CNN Attack	Success of Frequency Analysis
0	98%	92%
0.1	96%	26%
0.2	96%	5%
0.3	92%	0%

Table 8: Impact of flip ratio on frequency analysis attack.

Var-CNN attack and frequency analysis attack. When the flip ratio is ≥ 0.2, frequency analysis countermeasure offer no benefit against our defense.

Anomaly Detection. We also consider attackers that using traditional anomaly detection techniques to distinguish patches from normal packet sequences.

We compute 80 distinct patches using $\texttt{DF}_{\texttt{Sirinam}}$ for each of the 900 websites in `Rimmer`, for a total of 72,000 patches. We compare these patches to 72,000 natural packet sequences of the same length randomly chosen from original traces (with random offsets). *First*, we run a 2-means clustering on the features space of sequences (features extracted by $\texttt{DF}_{\texttt{Sirinam}}$). The resulting clusters contain 47% and 53% patches respectively and fail to distinguish patches from normal sequences. *Second*, we also try to separate patches using supervised training. We train a DF classifier on packet sequences of patches and natural packets. The classifier achieves 58% accuracy, only slightly outperforming random guessing.

7.2 Preprocessing Network Traces

A simple but effective approach to defeat adversarial patch is to transform the input before using them for both training and inferencing [12, 25]. In the case of WF attacks, we consider 3 possible transformations: i) add "noise" by randomly flipping packets, ii) add "noise" by randomly dropping packets, iii) truncating the network traces after the first N packets. Note that the attacker is processing the traces locally and does not modify any packets in network.

Our tests show that none of these transformations impact our defense in any meaningful way. Flipping random packets in the trace degrades the classification accuracy of the attacker classifier by 22%, but *Dolos* remains successful $>96\%$ (Table 5). Next, Table 5 shows randomly dropping packets from the trace decreases protection success rate by at most 2%, but more significantly, degrades attacker's classification accuracy (by 32%). Finally, truncating the trace degrades the attacker's classification accuracy but *Dolos* protection success remains $>94\%$ (Table 6).

7.3 More Robust Attack Classifiers

Next, we evaluate the feasibility of techniques to improve the robustness of attacker classifiers against adversarial patches.

Training on Multiple Patches. In §6, the attacker trained their classifier on traces protected by patch generated using a single secret, and failed to achieve high attack performance. Here, we consider a more general adversarial training approach, that trains the attacker's model against patched traces generated from *multiple* distinct secrets. Training against multiple targets gets the model closer to a more complete coverage of the space of potential adversarial patches.

The attacker uses *Dolos* source code (with $\texttt{DF}_{\texttt{Sirinam}}$ feature extractor) to generate defended traces using N randomly selected keys for each website in `Rimmer` and trains a Var-CNN classifier. On the defender side, the traces is protected using $\texttt{DF}_{\texttt{Sirinam}}$ feature extractor but a different key[5]. Figure 11 shows that as the attacker trains against more patched traces generated from different secrets, there is a small gain in robustness by the attacker's model. At its lowest point, the efficacy of *Dolos* patches drops to 87%. Across all of our countermeasures tested, this is the most effective.

Robust Attack Classifier. In adversarial patch training, a model is iteratively retrained on adversarial patches generated by the model. This technique is similar to but different from training on defended traces, which are generated by the defender's model. In adversarial patch training, the classifier is iteratively trained on adversarial patches generated on the attacker model such that the classifier is robust to any type of adversarial patches.

We evaluate *Dolos* against increasingly more robust classifiers on the attacker side. Model robustness directly correlates with the number of epochs trained using adversarial training [41,45,71,77]. In our experiment, the defender uses the $\texttt{DF}_{\texttt{Sirinam}}$ feature extractor (adversarially trained for 20 epochs) to generate patches. We test the defense against attack classifier (Var-CNN) with varying robustness (adversarially trained from 20 to 200 epochs). Figure 7 shows the protection success rate remains $\leq 94\%$ against all the robust classifiers and does not trend downwards as the model becomes more robust. This shows that generic adversarial training is less effective than training on defended traces, likely because the latter is more targeted towards specific types of perturbations generated by the defender.

Training Orthogonal Classifiers. Another countermeasure by the attacker can be to explicitly avoid the features used by the defender to generate patches, and to find other features for their trace classification. If successful, it would produce a classifier that is largely resistant to the patch. One approach is to build an attack classifier that has orthogonal feature space as the defender's feature extractor. The attacker adds an additional loss term to model training to minimize the neuron cosine similarity at an intermediate layer of the model. We train such a classifier using `Rimmer` and Var-CNN model architecture. In our tests, the classifier only achieves 8% normal classification accuracy after 20 epochs of training. This likely shows that there are not enough alternative, orthogonal features that can accurately identify destination websites from network traces.

Other Countermeasures Against Adversarial Patches. There are other defenses against adversarial patches explored in the computer vision domain. However, most of them are limited to small input perturbations (less than 5% of the input) [2,19,81]. Others only work on contiguous patches [29,47,49]. To the best of our knowledge, there exist no effective defense against larger patches (30% of input) or nonconsecutive adversarial patches induced by *Dolos*. While it is always possible the community will develop more effective defenses against larger adversarial patches, there exists a proven lower bound on adversarial robustness that increases as the size of perturbation increases [6]. Thus, it is difficult to be robust against large input perturbations without sacrificing classification accuracy.

[5]We do not consider the case where the user and attacker's secrets match, since its probability is extremely small (i.e., $\leq 50/400K$ in cases we tested).

8 Conclusion and Limitations

The primary contribution of *Dolos* is an effective defense against website fingerprinting attacks (both traditional ML based and DNN-based) that can run in real time to protect users. Our work is the first to apply the concept of adversarial patches to WF defenses.

However, there are questions we have yet to study in detail. First, while most recent defenses and attacks focus on a direction-only threat model and ignore information leakage through time gaps [21,31,34,58,65,75], some recent WF attacks [7,59] also utilize time gaps between packets to classify websites. We believe *Dolos* can be extended to also defend against attacks utilizing time-gaps, and plan on addressing this task in ongoing work. Second, we have not yet studied *Dolos* deployed in the wild. Real measurements and tests in the wild may reveal additional considerations, leading to additional fine tuning of our system design.

References

[1] ABE, K., AND GOTO, S. Fingerprinting attack on tor anonymity using deep learning. *APAN 42* (2016), 15–20.

[2] AKHTAR, N., LIU, J., AND MIAN, A. Defense against universal adversarial perturbations. In *Proc. of CVPR* (2018), pp. 3389–3398.

[3] https://www.alexa.com, 2017.

[4] BAGDASARYAN, E., AND SHMATIKOV, V. Blind backdoors in deep learning models. *arXiv preprint arXiv:2005.03823* (2020).

[5] BARKER, E., BARKER, E., BURR, W., POLK, W., SMID, M., ET AL. *Recommendation for key management: Part 1: General*. NIST, 2006.

[6] BHAGOJI, A. N., CULLINA, D., AND MITTAL, P. Lower bounds on adversarial robustness from optimal transport. In *Proc. of NeurIPS* (2019), pp. 7498–7510.

[7] BHAT, S., LU, D., KWON, A., AND DEVADAS, S. Var-cnn: A data-efficient website fingerprinting attack based on deep learning. *PoPETS 2019*, 4 (2019), 292–310.

[8] BROWN, T. B., MANÉ, D., ROY, A., ABADI, M., AND GILMER, J. Adversarial patch. *arXiv preprint arXiv:1712.09665* (2017).

[9] CAI, X., NITHYANAND, R., AND JOHNSON, R. Cs-buflo: A congestion sensitive website fingerprinting defense. In *Proc. of WPES* (2014), pp. 121–130.

[10] CAI, X., NITHYANAND, R., WANG, T., JOHNSON, R., AND GOLDBERG, I. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proc. of CCS* (2014), pp. 227–238.

[11] CAI, X., ZHANG, X. C., JOSHI, B., AND JOHNSON, R. Touching from a distance: Website fingerprinting attacks and defenses. In *Proc. of CCS* (2012), pp. 605–616.

[12] CARLINI, N., AND WAGNER, D. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proc. of AISec* (2017).

[13] CARLINI, N., AND WAGNER, D. Towards evaluating the robustness of neural networks. In *Proc. of IEEE S&P* (2017).

[14] CHARLES, Z., ROSENBERG, H., AND PAPAILIOPOULOS, D. A geometric perspective on the transferability of adversarial directions. In *Proc. of AISTAT* (2019), PMLR, pp. 1960–1968.

[15] CHEN, P.-Y., SHARMA, Y., ZHANG, H., YI, J., AND HSIEH, C.-J. Ead: elastic-net attacks to deep neural networks via adversarial examples. In *Proc. of AAAI* (2018).

[16] CHEN, S.-T., CORNELIUS, C., MARTIN, J., AND CHAU, D. H. P. Shapeshifter: Robust physical adversarial attack on faster r-cnn object detector. In *Proc. of ECML PKDD* (2018), Springer, pp. 52–68.

[17] CHERUBIN, G. Bayes, not naïve: Security bounds on website fingerprinting defenses. *PoPETS 2017*, 4 (2017), 215–231.

[18] CHERUBIN, G., HAYES, J., AND JUAREZ, M. Website fingerprinting defenses at the application layer. *PoPETS 2017*, 2 (2017), 186–203.

[19] CHIANG, P.-Y., NI, R., ABDELKADER, A., ZHU, C., STUDOR, C., AND GOLDSTEIN, T. Certified defenses for adversarial patches. *arXiv preprint arXiv:2003.06693* (2020).

[20] DANEZIS, G. Statistical disclosure attacks. In *Proc. of IFIP SEC* (2003), Springer, pp. 421–426.

[21] DE LA CADENA, W., MITSEVA, A., HILLER, J., PENNEKAMP, J., REUTER, S., FILTER, J., ENGEL, T., WEHRLE, K., AND PANCHENKO, A. Trafficsliver: Fighting website fingerprinting attacks with traffic splitting. In *Proc. of CCS* (2020), pp. 1971–1985.

[22] DEMONTIS, A., MELIS, M., PINTOR, M., JAGIELSKI, M., BIGGIO, B., OPREA, A., NITA-ROTARU, C., AND ROLI, F. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *Proc. of USENIX Security* (2019), pp. 321–338.

[23] 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 *Proc. of IEEE S&P* (2012), IEEE, pp. 332–346.

[24] EBRAHIMI, J., RAO, A., LOWD, D., AND DOU, D. Hotflip: White-box adversarial examples for text classification. *arXiv preprint arXiv:1712.06751* (2017).

[25] FEINMAN, R., CURTIN, R. R., SHINTRE, S., AND GARDNER, A. B. Detecting adversarial samples from artifacts. *arXiv:1703.00410* (2017).

[26] GONG, J., AND WANG, T. Zero-delay lightweight defenses against website fingerprinting. In *Proc. of USENIX Security* (2020), pp. 717–734.

[27] GOODFELLOW, I. J., SHLENS, J., AND SZEGEDY, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[28] GROSSE, K., PAPERNOT, N., MANOHARAN, P., BACKES, M., AND MCDANIEL, P. Adversarial examples for malware detection. In *Proc. of ESORICS* (2017), Springer, pp. 62–79.

[29] HAYES, J. On visible adversarial perturbations & digital watermarking. In *Proc. of CVPR* (2018), pp. 1597–1604.

[30] HAYES, J., AND DANEZIS, G. k-fingerprinting: A robust scalable website fingerprinting technique. In *Proc. of USENIX Security* (2016), pp. 1187–1203.

[31] HENRI, S., GARCIA-AVILES, G., SERRANO, P., BANCHS, A., AND THIRAN, P. Protecting against website fingerprinting with multihoming. *PoPETS 2020*, 2 (2020), 89–110.

[32] HERRMANN, D., WENDOLSKY, R., AND FEDERRATH, H. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proc. of CCSW* (2009), pp. 31–42.

[33] HOLLAND, J. K., AND HOPPER, N. Regulator: A powerful website fingerprinting defense. *arXiv preprint arXiv:2012.06609* (2020).

[34] HOU, C., GOU, G., SHI, J., FU, P., AND XIONG, G. Wf-gan: Fighting back against website fingerprinting attack using adversarial learning. In *Proc. of ISCC* (2020), IEEE, pp. 1–7.

[35] ILYAS, A., SANTURKAR, S., TSIPRAS, D., ENGSTROM, L., TRAN, B., AND MADRY, A. Adversarial examples are not bugs, they are features. In *Proc. of NeurIPS* (2019).

[36] JUAREZ, M., IMANI, M., PERRY, M., DIAZ, C., AND WRIGHT, M. Toward an efficient website fingerprinting defense. In *Proc. of ESORICS* (2016), Springer, pp. 27–46.

[37] KARMON, D., ZORAN, D., AND GOLDBERG, Y. Lavan: Localized and visible adversarial noise. In *Proc. of ICML* (2018), PMLR, pp. 2507–2515.

[38] KOLOSNJAJI, B., DEMONTIS, A., BIGGIO, B., MAIORCA, D., GIACINTO, G., ECKERT, C., AND ROLI, F. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *Proc. of EUSIPCO* (2018), IEEE, pp. 533–537.

[39] KURAKIN, A., GOODFELLOW, I., AND BENGIO, S. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533* (2016).

[40] LEE, J. *A first course in combinatorial optimization*, vol. 36. Cambridge University Press, 2004.

[41] LI, B., WANG, S., JANA, S., AND CARIN, L. Towards understanding fast adversarial training. *arXiv preprint arXiv:2006.03089* (2020).

[42] Li, S., Guo, H., and Hopper, N. Measuring information leakage in website fingerprinting attacks and defenses. In *Proc. of CCS* (2018), pp. 1977–1992.

[43] Liberatore, M., and Levine, B. N. Inferring the source of encrypted http connections. In *Proc. of CCS* (2006), pp. 255–263.

[44] Lu, L., Chang, E.-C., and Chan, M. C. Website fingerprinting and identification using ordered feature sequences. In *Proc. of ESORICS* (2010), Springer, pp. 199–214.

[45] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. In *Proc. of ICLR* (2018).

[46] Mallesh, N., and Wright, M. An analysis of the statistical disclosure attack and receiver-bound cover. *Computers & Security 30*, 8 (2011), 597–612.

[47] McCoyd, M., Park, W., Chen, S., Shah, N., Roggenkemper, R., Hwang, M., Liu, J. X., and Wagner, D. Minority reports defense: Defending against adversarial patches. *arXiv preprint arXiv:2004.13799* (2020).

[48] Moosavi-Dezfooli, S.-M., Fawzi, A., Fawzi, O., and Frossard, P. Universal adversarial perturbations. In *Proc. of CVPR* (2017), pp. 1765–1773.

[49] Naseer, M., Khan, S., and Porikli, F. Local gradients smoothing: Defense against localized adversarial attacks. In *Proc. of WACV* (2019), pp. 1300–1307.

[50] Nasr, M., Bahramali, A., and Houmansadr, A. Blind adversarial network perturbations. *arXiv preprint arXiv:2002.06495* (2020).

[51] Nithyanand, R., Cai, X., and Johnson, R. Glove: A bespoke website fingerprinting defense. In *Proc. of WPES* (2014), pp. 131–134.

[52] Panchenko, A., Lanze, F., Pennekamp, J., Engel, T., Zinnen, A., Henze, M., and Wehrle, K. Website fingerprinting at internet scale. In *Proc. of NDSS* (2016).

[53] Panchenko, A., Niessen, L., Zinnen, A., and Engel, T. Website fingerprinting in onion routing based anonymization networks. In *Proc. of WPES* (2011), pp. 103–114.

[54] Papernot, N., McDaniel, P., Swami, A., and Harang, R. Crafting adversarial input sequences for recurrent neural networks. In *Proc. of MILCOM* (2016), IEEE, pp. 49–54.

[55] Perry, M. Tor protocol specification proposal, 2015. https://gitweb.torproject.org/torspec.git/tree/proposals/254-padding-negotiation.txt.

[56] Petrov, D., and Hospedales, T. M. Measuring the transferability of adversarial examples. *arXiv preprint arXiv:1907.06291* (2019).

[57] Pydi, M. S., and Jog, V. Adversarial risk via optimal transport and optimal couplings. In *Proc. of ICML* (2020), pp. 7814–7823.

[58] Rahman, M. S., Imani, M., Mathews, N., and Wright, M. Mockingbird: Defending against deep-learning-based website fingerprinting attacks with adversarial traces. *TIFS* (2020), 1594–1609.

[59] Rahman, M. S., Sirinam, P., Mathews, N., Gangadhara, K. G., and Wright, M. Tik-tok: The utility of packet timing in website fingerprinting attacks. *PoPETS 2020*, 3 (2020), 5–24.

[60] Rao, S., Stutz, D., and Schiele, B. Adversarial training against location-optimized adversarial patches. *arXiv preprint arXiv:2005.02313* (2020).

[61] Ren, S., Deng, Y., He, K., and Che, W. Generating natural language adversarial examples through probability weighted word saliency. In *Proc. of ACL* (2019), pp. 1085–1097.

[62] Rimmer, V., Preuveneers, D., Juarez, M., Van Goethem, T., and Joosen, W. Automated website fingerprinting through deep learning. In *Proc. of NDSS* (2018).

[63] Shafahi, A., Huang, W. R., Studer, C., Feizi, S., and Goldstein, T. Are adversarial examples inevitable? In *ICLR* (2019).

[64] Shalev-Shwartz, S., and Ben-David, S. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[65] Sirinam, P., Imani, M., Juarez, M., and Wright, M. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proc. of CCS* (2018), pp. 1928–1943.

[66] Sirinam, P., Mathews, N., Rahman, M. S., and Wright, M. Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning. In *Proc. of CCS* (2019), pp. 1131–1148.

[67] Song, D., Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Tramer, F., Prakash, A., and Kohno, T. Physical adversarial examples for object detectors. In *Proc. of WOOT* (2018).

[68] Suciu, O., Coull, S. E., and Johns, J. Exploring adversarial examples in malware detection. In *Proc. of SPW* (2019), IEEE, pp. 8–14.

[69] Suciu, O., Mărginean, R., Kaya, Y., Daumé III, H., and Dumitraş, T. When does machine learning fail? generalized transferability for evasion and poisoning attacks. In *Proc. of USENIX Security* (2018).

[70] Sun, Q., Simon, D. R., Wang, Y.-M., Russell, W., Padmanabhan, V. N., and Qiu, L. Statistical identification of encrypted web browsing traffic. In *Proc. of IEEE S&P* (2002), IEEE, pp. 19–30.

[71] Tramer, F., and Boneh, D. Adversarial training and robustness for multiple perturbations. In *Proc. of NeurIPS* (2019), pp. 5866–5876.

[72] Uesato, J., O'Donoghue, B., Oord, A. v. d., and Kohli, P. Adversarial risk and the dangers of evaluating against weak attacks. *arXiv preprint arXiv:1802.05666* (2018).

[73] Wallace, E., Feng, S., Kandpal, N., Gardner, M., and Singh, S. Universal adversarial triggers for attacking and analyzing nlp. *arXiv preprint arXiv:1908.07125* (2019).

[74] WANG, T., CAI, X., NITHYANAND, R., JOHNSON, R., AND GOLDBERG, I. Effective attacks and provable defenses for website fingerprinting. In *Proc. of USENIX Security* (2014), pp. 143–157.

[75] WANG, T., AND GOLDBERG, I. Walkie-talkie: An efficient defense against passive website fingerprinting attacks. In *Proc. of USENIX Security* (2017), pp. 1375–1390.

[76] WANG, X., JIN, H., AND HE, K. Natural language adversarial attacks and defenses in word level. *arXiv preprint arXiv:1909.06723* (2019).

[77] WONG, E., RICE, L., AND KOLTER, J. Z. Fast is better than free: Revisiting adversarial training. In *Proc. of ICLR* (2020).

[78] WRIGHT, M. K., ADLER, M., LEVINE, B. N., AND SHIELDS, C. The predecessor attack: An analysis of a threat to anonymous communications systems. *TISSEC 7*, 4 (2004), 489–522.

[79] WRIGHT, M. K., ADLER, M., LEVINE, B. N., AND SHIELDS, C. Passive-logging attacks against anonymous communications systems. *TISSEC 11*, 2 (2008), 1–34.

[80] WU, Z., LIM, S.-N., DAVIS, L., AND GOLDSTEIN, T. Making an invisibility cloak: Real world adversarial attacks on object detectors. *arXiv preprint arXiv:1910.14667* (2019).

[81] XIANG, C., BHAGOJI, A. N., SEHWAG, V., AND MITTAL, P. Patchguard: Provable defense against adversarial patches using masks on small receptive fields. *arXiv preprint arXiv:2005.10884* (2020).

[82] YOSINSKI, J., CLUNE, J., BENGIO, Y., AND LIPSON, H. How transferable are features in deep neural networks? In *Proc. of NeurIPS* (2014).

[83] ZHANG, W. E., SHENG, Q. Z., ALHAZMI, A., AND LI, C. Adversarial attacks on deep-learning models in natural language processing: A survey. *TIST 11*, 3 (2020), 1–41.

A Effectiveness against Intersection Attacks

Intersection attacks are popular attacks against anonymity systems [20,46,78,79]. In the context of website fingerprinting, an intersection attacker assumes the victim visits the same websites regularly and monitors the victim's network traffic for a longer time period (e.g., every day over multiple days). Using the additional information, the attacker is able to make better inferences on user's traces. We test *Dolos* against an intersection attack intersection attack used by a previous defense [58].

For each of the victim's browsing traces, the attacker logs the top-k results of the attack classifier (top-k is the k output websites that have the highest probabilities that the trace belongs to). If a website consistently appear in the top-k results, then the attacker may believe that this site is in fact the website that the user is visiting. We choose the same attack setup as [58]. We assume attacker observes 5 separate visits to the same website. Then the attacker saves the top-10 labels predicted by the classifier (using Var-CNN attack). The attack is successful if the selected label is the most frequently appeared label within joint list of 50 labels (5 sets of 10 top-10 labels). We test on 20 randomly selected websites from `Rimmer`. For all cases, the frequency of the correct website is far from the most frequent one (in the best case it ranks number 4 out of 41 websites) and the correct website never appears in all 5 rounds. Thus, we conclude that intersection attacks is not effective against *Dolos*.

B Theoretical Justification of Defense

We show that *Dolos* provides provable robustness guarantees when both the attacker and defender use the same fixed feature extractor Φ.

We use recent theoretical results [6,57] on learning 2-class classifiers in the presence of adversarial examples which has shown that as the strength of the attacker (defender in our case) increases, the $0-1$ loss incurred by *any classifier* is lower bounded by a transportation cost between the conditional distributions of the two classes. In the case of image classification, which is the example considered in previous work, the budget is typically too small to observe interesting behavior in terms of this lower bound.

However, since we consider network traffic traces to which much larger amounts of perturbation can be added, we encounter non-trivial regimes of this bound. This implies that with a sufficiently large bandwidth, no classifier used by the attacker will be able to distinguish between traces from the source and target classes. In order to demonstrate this, we make the following assumptions:

1. The attacker is attempting to distinguish if a trace x belongs to the original class W or the target class T, with distributions P_W and P_T respectively, both of which may be defended (i.e. adversarially perturbed).
2. The attacker uses a classifier function F acting over the feature space $\Phi(\mathcal{X})$, where F can be any measurable function and $\Phi : \mathcal{X} \rightarrow \mathbb{R}^k$ is any fixed feature extractor. The resulting end-to-end classifer is represented by the tuple (Φ, F).
3. The defender has an ℓ_2 norm perturbation budget of ε_Φ in the feature space, matching the choice of $\text{Dist}(\cdot,\cdot)$ in Eq. 1. The feature space budget is related to the input space bandwidth overhead R by a mapping function M that maps balls of radius R in the input space to balls of radius at least ε_Φ in the feature space.

Given these assumptions, we can now state the following theorem, adapted from Bhagoji et al. [6]:

Theorem 1 (Upper bound on attacker success)**.** *For any pair of classes W and T with conditional probability distributions P_W and P_T, and the joint distribution P, over X, and with a fixed feature extractor Φ, the attack success rate of any classifier F is*

$$ASR((\Phi,F),P,R) \leq \frac{1}{2}\left(1 + C_{\varepsilon_\Phi}(\Phi(P_W),\Phi(P_T))\right). \quad (3)$$

Proof. The end-to-end classifier has a fixed feature extractor Φ and a classifier function F that can be optimized over. To determine an upper bound on the attack success rate achievable by a classifier F, we have

$$\begin{aligned} ASR((F,\Phi),P,R) &= \mathbb{E}_{x\sim P}\left[\min_{|\tilde{x}-x|\leq R} \mathbf{1}\left((F,\Phi)(\tilde{x}) = y\right)\right], \\ &\leq \mathbb{E}_{\Phi(x)\sim\phi(P)}\left[\min_{||\Phi(\tilde{x})-\Phi(x)||_2\leq\varepsilon_\Phi} \mathbf{1}\left(F(\Phi(\tilde{x})) = y\right)\right], \\ &= ASR(F,\Phi(P),\varepsilon_\Phi) \end{aligned}$$

The $\leq$ arises from a conservative estimate of the distance moved in feature space. Having transformed the attack success rate calculation to one over the feature space, we can now directly apply Theorem 1 from [6], which gives

$$\max_F ASR(F,\Phi(P),\varepsilon_\Phi) = \frac{1}{2}\left(1 + C_{\varepsilon_\Phi}(\Phi(P_W),\Phi(P_T))\right).$$

From [6],

$$\begin{aligned} & C(\Phi(P_W),\Phi(P_T)) \\ &= \inf_{P_{WT}\in\Pi(P_W,P_T)} \mathbb{E}_{(x_W,x_T)\sim P_{WT}}\left[\mathbf{1}(|\Phi(x_W)-\Phi(x_T)| \geq 2\varepsilon_\Phi)\right], \end{aligned} \quad (4)$$

where $\Pi(P_W,P_T)$ is the set of joint distributions over $\mathcal{X}_W \times \mathcal{X}_T$ with marginals P_W and P_T. ∎

Dataset	Defender's Feature Extractor	Protection Success Rate Against WF Attacks				
		k-NN	k-FP	CUMUL	DF	Var-CNN
`Sirinam`	$\texttt{DF}_{\texttt{Sirinam}}$	96%	97%	93%	95%	91%
	$\texttt{DF}_{\texttt{Rimmer}}$	96%	94%	97%	92%	93%
	$\texttt{VarCNN}_{\texttt{Sirinam}}$	94%	92%	95%	93%	96%
	$\texttt{VarCNN}_{\texttt{Rimmer}}$	97%	96%	95%	95%	94%
`Rimmer`	$\texttt{DF}_{\texttt{Sirinam}}$	94%	92%	95%	93%	92%
	$\texttt{DF}_{\texttt{Rimmer}}$	95%	94%	96%	97%	91%
	$\texttt{VarCNN}_{\texttt{Sirinam}}$	94%	97%	98%	94%	92%
	$\texttt{VarCNN}_{\texttt{Rimmer}}$	96%	95%	96%	95%	97%

Table 9: Protection performance of *Dolos* using non-robust feature extractor against different WF attacks when transfering to classifiers trained on different datasets and/or architecture.

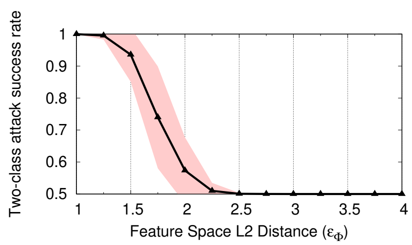

Figure 12: Upper bound on the effectiveness of *any attack classifier* for a fixed feature extractor Φ, averaged over 500 choices of source-pairs.

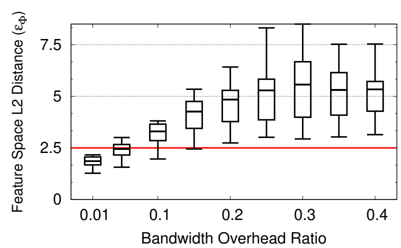

Figure 13: Variation in the distance moved in feature space with the input bandwidth overhead, averaged over a 100 different targets for a fixed source.

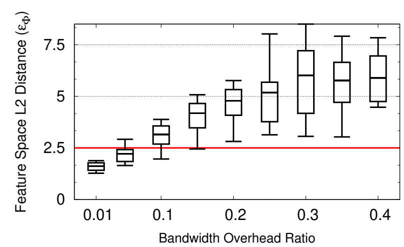

Figure 14: Variation in the distance moved in feature space with the input bandwidth variation, averaged over a single target for 20 different sources.

The main takeaway from the above theorem is that the better separated the perturbed feature vectors from the two classes are, the higher the attack success rate will be. Thus, from the defender's perspective, the bandwidth R has to be sufficient to ensure that the resulting ε_Φ leads to low separability.

Empirical upper bounds. We compute the feature space distances between 500 different sets of source W and target T websites and plot the maximum attack success rate as the budget ε_Φ in the feature space is varied (Figure 12). We use a robust feature extractor trained on the `Rimmer`dataset to derive this upper bound. With a feature space budget of 2.5, the attack success rate drops to 50%, which for a two-class classification problem implies that no classifier can distinguish between the two classes.

Now, it remains to be established that a reasonable input bandwidth overhead can lead to a feature space budget of 2.5. In Figures 13 and 14, we see that varying over 20 choices of sources with respect to a fixed target and 100 targets for a fixed source, the minimum distance moved in the feature space is larger than 2.5, making it a conservative estimate for ε_Φ in Theorem 1.

Remarks We note that our analysis here is restricted to the 2-class setting, thus the conclusions drawn may not apply for all source-target pairs. We hypothesize that this explains the lower than 100% protection rate at $R = 0.3$ we observe in §6, since some pairs of classes may not be sufficiently well-separated in feature space. We also note that the analysis used here improves over that of Cherubin [17], where the effectiveness of an attack is only tested against a fixed defense, while we provide optimal bounds in the setting where the attacker and defender can adapt to one another.