POSTER: Adversarial Traces for Website Fingerprinting Defense

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

Mohammad Saidur Rahman
Rochester Institute of Technology
saidur.rahman@mail.rit.edu

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

ABSTRACT

Website Fingerprinting (WF) is a traffic analysis attack that enables an eavesdropper to infer the victim's web activity even when encrypted and even when using the Tor anonymity system. Using deep learning classifiers, the attack can reach up to 98% accuracy. Existing WF defenses are either too expensive in terms of bandwidth and latency overheads (e.g. 2-3 times as large or slow) or ineffective against the latest attacks. In this work, we explore a novel defense based on the idea of *adversarial examples* that have been shown to undermine machine learning classifiers in other domains. Our *Adversarial Traces* defense adds padding to a Tor traffic trace in a manner that reliably fools the classifier into classifying it as coming from a different site. The technique drops the accuracy of the state-of-the-art attack from 98% to 60%, while incurring a reasonable 47% bandwidth overhead, showing its promise as a possible defense for Tor.

KEYWORDS

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

1 INTRODUCTION

Tor is known to be vulnerable to traffic analysis attacks. An adversary who observes the both entry and exit sides of the traffic on a Tor connection is able to correlate the traffic and link the client to her destination. An adversary needs significant resources to perform this attack reliably. A branch of traffic analysis that requires fewer resources is *Website Fingerprinting* (WF). The goal of the WF adversary is to identify which websites the client is visiting by observing only the connection between the client and the guard, as shown in Figure 1. 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.

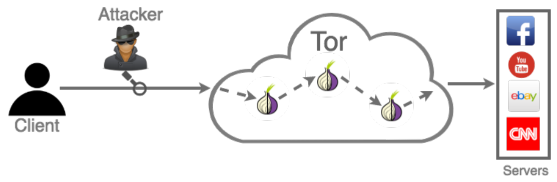


Figure 1: Website Fingerprinting Attack Model.

The WF attack is a supervised classification problem, in which the websites are the labels and each traffic trace is an instance to be classified or trained on. The accuracy rate of the state-of-the-art WF attack is 98% in a *closed-world* test [8].

In response to the threat of WF attacks, there have been several defenses proposed [1, 2, 5–7, 11]. 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 delay some packets. The BuFLO family of defenses (including BuFLO [3], CS-BuFLO [1], and Tamaraw [2]) apply these techniques and are effective, but make loading a website take two or three times as long as in Tor. WTF-PAD[4] offers lower overheads, but Sirinam et al. show an attack that reaches 90% accuracy against it. The more recently proposed Walkie-Talkie [10] is both effective and efficient, but there are major challenges to practical deployment.

In this work, we introduce a new defense strategy using adversarial examples generated by a deep neural network. We propose a new method to modify the website traces that causes misclassification in the classifier with moderate amounts of bandwidth, even if the attacker is trained on the traces. Our defense drops the accuracy rate of state-of-the-art attack from 98% to 60% with 47% bandwidth overhead.

2 A NEW WF DEFENSE

We now introduce a new mechanism to perturb traffic traces such that the classifier is not able to identify them reliably. We adapt the idea of *targeted* adversarial examples [9]. To defend a given trace (the *source sample*), our technique randomly picks a target sample and gradually changes the source sample in a direction to get closer to the target sample. Eventually, the sample has moved enough to cause the classifier to misclassify it.

More concretely, assume that we have a set of sensitive sites $\mathcal{S}$ that we want to protect and a model $f(x)$ (called *detector*) that is trained on a set of data from $\mathcal{S}$ (we will later discuss the cases whether $f(x)$ should be trained on only sensitive sites or both sensitive and non-sensitive sites). We consider traffic trace I_s as an instance of source class $s \in \mathcal{S}$ that we want to alter such that it is classified to target class t, $t = f(I_s)$ and $t \neq s$. I_s is a sequence of the bursts, $I_s = \left[b_0^I, b_1^I, ..., b_n^I\right]$. 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 increase the volume of the bursts by sending dummy packets. If $\delta_i < 0$, it means that we should drop some packets to reduce the burst volume, but dropping real packets means losing data.

To protect source sample I_s, we pick p random samples from other classes, $P_{I_s} = \left[I^0{}_{T_0}, I^1{}_{T_1}, ..., I^p{}_{T_m}, \right]$. P_{I_s} is the *target pool* for I_s. $I^j{}_{T_i}$ is the j-th sample in the target pool and belongs to target class $T_i \neq s$. We want to pick a target class and re-cast the source sample to be classified as that target class. To decrease amount of change to the source sample, since adding padding adds bandwidth overhead, we pick the sample from the target pool that is closest to the source sample. We define closeness using the l_2 norm distance. Formally:

$$D(x, y) = l_2(x - y)$$
$$I_T = \underset{I_t \in P_{I_s}}{\operatorname{argmin}} D(I_s, I_t)$$

Then we modify the source sample to move toward this target sample.

Our goal is to increase the volumes of selected bursts in the source sample such that the source sample is not classified as class s and the amount of change is as small as possible to minimize the bandwidth overhead. To make the source sample to leave the source class, we move toward the nearest sample (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}.

$$\Delta = [\delta_0, \delta_1, \cdots, \delta_n] \quad (\forall i \in [0, \cdots, n] : \delta_i \geqslant 0)$$
$$I_s^{new} = I_s + \Delta$$

To find Δ that minimizes overhead, we should minimize distance $D(I_s^{new}, I_T)$. To do this, we compute the gradient of the distance with respect to the input. The gradient points in the direction of steepest ascent, which would maximize the distance. Therefore, we compute the gradient of the negative of the distance with respect to the input, and we move the source sample that direction towards the target sample. In particular:

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

where b_i is the i-th burst in input I. To modify the source sample, we change bursts such that their corresponding values in $(-D(I, I_T))$ are positive. Our perturbation vector Δ is:

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

where α is paramter that amplifies the output of the gradient. The choice of α has an impact on the convergence and the bandwidth overhead. If we pick large value for α, we will take bigger steps toward the target sample and we will add more overhead. We modify the source sample by summing it with Δ, ($I_s^{new} = I_s + \Delta$). We iterate this process, computing Δ for each I_s and updating the source sample until we leave the source class, $f(I_s^{new}) \neq s$ or the number of iterations passes the maximum allowed iterations. In our experiments, we set this maximum as 200 iterations.

Because we only increase 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 (we used threshold 0.001) for a few iterations (we used 10 iterations), and we are still in the source class, we refill the pool with new samples and pick a new target sample I_T to continue the process.

3 EVALUATIONS

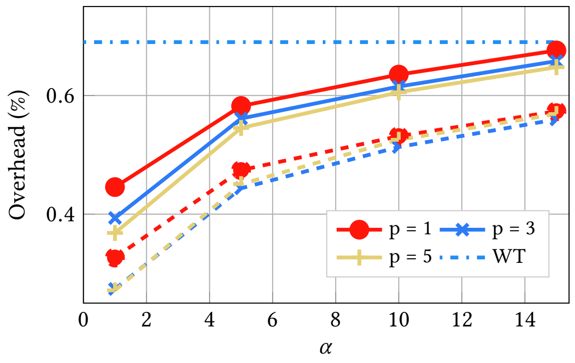


Figure 2: Bandwidth Overhead: the bandwidth overhead of generated samples as α and target pool size vary. Dashed lines show the results of Case I and solid lines show the results of Case II.

In our evaluation, we break the data into two non-overlapping sets: the Attacker Set and the Defender Set. Each set has a *monitored set* of 83 classes, each representing a website of interest to the attacker, with 360 instances each. Moreover, our dataset contains an *unmonitored set* of 40,000 instances from 40,000 different sites, one instance per site.

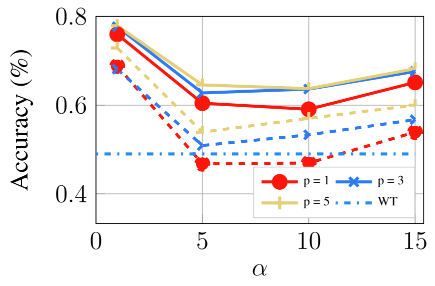
We examine the bandwidth overhead and reduction in attacker accuracy of traces protected by our method. We use traces in the training data and generated their defended forms by the method described in the previous section. We first require a *detector* ($f(x)$) to identify when the generated samples leave their source class. Thus, we define the *detector*, a CNN model, and train it on the traces in Attacker Set. In our evaluations we examine two cases:

- Case I: We fill the target pool with instances from the Attacker Set. In this case, the *detector* has been trained on the target classes.
- Case II: We fill the target pool with instances from the unmonitored. In this case, the *detector* has not been trained on the target samples.

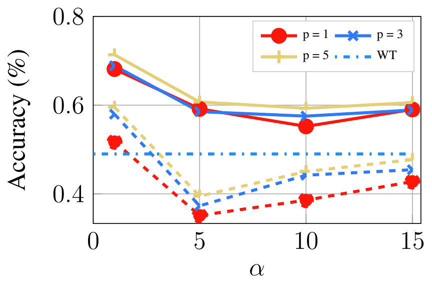
We generated defended samples with various settings. We varied α and p to evaluate their effect on the strength of the defended traces and the overhead. We measured the detectability of the defended samples by applying the DF attack [8] on them. Sirinam el al. [8] suggest using 5,000 packets. Because both Walkie-Talkie and our method increase the size of the bursts, the number of packets in the traces increases. We thus use an input size of 10,000 packets, which is the 80th percentile of packet sequence lengths in our defended traces.

Figure 2 shows the bandwidth overhead in both Walkie-Talkie (WT) and our method for Case I (solid lines) and II (dashed lines) as α and p vary. As shown in the figure, as we increase α, the bandwidth overhead increases. Larger α values create longer steps toward the target samples and less fine-grained searches for an effective stopping point. Using a larger target pool moderately decreases bandwidth overhead in most cases.

Case I leads to lower bandwidth overhead compared to Case II. Therefore, picking target samples from classes that the *detector* has



(a) Case I



(b) Case II

Figure 3: Accuracy: the accuracy rate of the generated samples against the DF attack. Dashed lines depict the cases when the input size to the DF attack is 5,000 packets and solid lines show the results when it is 10,000 packets.

been trained on will drop the overhead. In all the evaluated settings, the bandwidth overhead of our method is lower than that of Walkie-Talkie. For α = 1 and p = 5 in Case I, the bandwidth overhead of our defense is 27%, 60% lower than for Walkie-Talkie. On the other end, for α = 15 and p = 5 in Case I, the bandwidth overhead of our defense is 56%, 18% lower than for Walkie-Talkie.

Figure 3 depicts the accuracy rate of the DF attack as α and p vary, for input sizes of 5,000 packets and 10,000 packets. Figure 3a and 3b depict the results of the evaluations in Case I and Case II, respectively. As α increases in both cases, the accuracy rate drops to its minimum and then slightly increases. On the other hand, increasing α raises the bandwidth overhead monotonically. Raising target pool size p can moderately increase the attacker's accuracy. According to Figure 3a, the lowest accuracy rate is 59% when α = 10 and p = 1, and its corresponding bandwidth overhead is 53%. α = 10 and p = 1 in Case II provides the lowest accuracy rate (55%) with bandwidth overhead of 63%.

Our evaluations show that Case I provides lower bandwidth overhead than Case II (between 15% to 27% lower) and the detectability of the generated samples is comparable with Case II. This means that picking target samples from classes that the *detector* trained on reduces bandwidth overhead. According to our results, our best setting is to pick target samples from the classes that the *detector* trained on with p=1 and α=5. Walkie-Talkie still has lower attack accuracy than this setting, 49% to our 60%, but its bandwidth overhead is 46% higher than our defense.

4 CONCLUSION & FUTURE WORK

In this work, we propose a new defense against WF attacks with lower bandwidth overhead than Walkie-Talkie, the state-of-the-art defense, with reasonable reductions in attacker accuracy. The defense uses a novel mechanism that adapts techniques used to create adversarial examples against machine learning classifiers, applying them to website traffic traces. The generated adversarial traces can limit the adversary even though he is trained on the adversarial traces. To protect a traffic trace, we add fake packets to the source trace to shorten the distance between the source sample and a randomly selected target sample representing another website. Our defense mechanism results in 47% bandwidth overhead and drops the accuracy rate of the state-of-the-art WF attack from 98% to 60%. We emphasize that our tests are conducted in the closed-world setting, where the attacker knows that the user is visiting one of the monitored set of websites. In the more realistic *open-world* setting, where the user could visit any site on the Web, 60% accuracy is very likely to lead to many false positives for the attacker. In future work, we plan to investigate more to improve the defense and show how to implement it.

ACKNOWLEDGMENT

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

REFERENCES

[1] 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)*.
[2] 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)*.
[3] 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)*.
[4] 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)*.
[5] 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)*.
[6] Rishab Nithyanand, Xiang Cai, and Rob Johnson. 2014. Glove: A Bespoke Website Fingerprinting Defense. In *Workshop on Privacy in the Electronic Society (WPES)*.
[7] Mike Perry. 2011. Experimental Defense for Website Traffic Fingerprinting. Tor Project Blog. https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting.
[8] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. 2018. Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning. arXiv:arXiv:1801.02265
[9] 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)*.
[10] Tao Wang and Ian Goldberg. 2017. Walkie-talkie: An efficient defense against passive website fingerprinting attacks. In *USENIX Security Symposium*.
[11] 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)*.