Poster: Zero-delay Lightweight Defenses against Website Fingerprinting

Jiajun GONG
Department of Computer Science and Engineering
Hong Kong University of Science and Technology
Hong Kong, China
jgongac@connect.ust.hk

Tao WANG
Department of Computer Science and Engineering
Hong Kong University of Science and Technology
Hong Kong, China
taow@cse.ust.hk

***Abstract*—Website Fingerprinting (WF) attacks threaten user privacy on anonymity networks because they can be used by network surveillants to identify the webpage being visited by extracting features from network traffic. A number of defenses have been put forward to mitigate the threat of WF, but they are limited: some have been defeated by stronger WF attacks, some are too expensive in overhead, while others are impractical to deploy.**

In this work, we propose two novel zero-delay lightweight defenses, FRONT and GLUE. We find that WF attacks rely on the feature-rich trace front, so FRONT focuses on obfuscating the trace front with dummy packets. It also randomizes the number and distribution of dummy packets for trace-to-trace randomness to impede the attacker's learning process. GLUE adds dummy packets between separate traces so that they appear to the attacker as a long consecutive trace, rendering the attacker unable to find their start or end points, let alone classify them. Our experiments show that with only 18% data overhead, our first defenses FRONT outperforms the best known lightweight defense, WTF-PAD, which has a 50% data overhead. With around 10%–50% data overhead, our second defense GLUE can lower the accuracy and precision of the best WF attacks to a degree comparable with the best heavyweight defenses.

***Index Terms*—Network Privacy, Network Traffic analysis, Website Fingerprinting**

I. Problem Description

A. Overview

Tor, an anonymity network based on onion routing [1], has become one of the most popular privacy enhancing technologies. However, it is shown to be vulnerable to *Website Fingerprinting* (WF), a kind of traffic analysis attack where a local attacker passively eavesdrops on network traffic to find out which webpage a client is visiting. WF succeeds by observing packet patterns such as the number of outgoing and incoming packets, packet rates, packet timing, and the ordering of packets. (WF attacks do not need to break encryption.) Hence, our goal is to design a defense such that it efficiently weakens WF attacks with low data overhead (i.e., adding few dummy packets) and zero latency overhead (i.e., delaying true packets).

B. Threat Model

We assume an WF attacker sits between a Tor user and the Tor network. The attacker only passively monitors user's network traffic without modifying, dropping or delaying any packet to find out which webpage is being visited. Anyone local to the user could be a potential WF attacker, such as ISP or even the entry node of the Tor network.

TABLE I
Defense parameters and variables in FRONT.

	Notation	Parameter
Parameters	N_c	Client's padding budget
	N_s	Proxy's padding budget
	W_{min}	Minimum padding time
	W_{max}	Maximum padding time
Variables	$n_c \leftarrow \bar{U}(1, N_c)$	Number of outgoing dummy packets
	$n_s \leftarrow \bar{U}(1, N_s)$	Number of incoming dummy packets
	$w_c \leftarrow U(W_{min}, W_{max})$	Client's padding window
	$w_s \leftarrow U(W_{min}, W_{max})$	Proxy's padding window

II. Defense Description

A. FRONT

The first zero-delay defenses we propose is called FRONT, **F**ront **R**andomized **O**bfuscation of **N**etwork **T**raffic.

1) Intuition:

- **Obfuscating feature-rich trace fronts**. The first few seconds of a trace (i.e., the trace front) leaks the most useful features for WF classification. That is because the trace front corresponds to loading HTML of a webpage and it is usually very unique. Thus we dedicate most of our data budget to obfuscating the trace front, instead of spreading them evenly over the trace.
- **Trace-to-trace randomness**. FRONT adds dummy packets in a highly random manner, ensuring that different traces of the same webpage look different to each other in total length, packet ordering, and packet directions. To do so, it randomizes the data budget and the region where we inject dummy packets.

2) Defense Design: There are three steps in using FRONT to defend a trace: sample a number of dummy packets, sample a padding window size and schedule dummy packets. Its parameters are summarized in Table I.

a) Sample a number of dummy packets: N_c and N_s are two parameters determining the data overhead of FRONT, respectively representing the client's padding budget and the proxy's padding budget. For each trace, the client samples n_c from the discretized uniform distribution between 1 and N_c, denoted as $\bar{U}(1, N_c)$; the proxy samples n_s from $\bar{U}(1, N_s)$.

n_c and n_s are the actual number of dummy packets they will inject into that trace.

b) Sample a padding window: FRONT spends most of its budget obfuscating trace fronts. To do so, both client and proxy will first generate a *padding window*, the time interval where most dummy packets are expected to be injected into the original trace. For each trace, the client samples w_c from the uniform distribution between W_{min} and W_{max}, denoted as $U(W_{min}, W_{max})$; the proxy samples w_s from the same distribution. The reason we set a lower bound W_{min}, instead of 0, is to ensure that the generated padding window size is not too small; if it is too small, the defense may require an extreme bandwidth rate to support.

c) Schedule dummy packets: After sampling the above variables, the client and proxy generate separate timetables to schedule when their respective n_c and n_s dummy packets will be sent. They generate the timestamps by sampling n_c and n_s times from a negative exponential distribution. Its probability density function is:

$$f(t; w) = \begin{cases} \frac{2}{w} e^{-\frac{2}{w}t} & t \geq 0 \\ 0 & t < 0 \end{cases},$$

where w is w_c for the client and w_s for the proxy. Both client and proxy send true packets with no delays and send dummy packets according to their own timetable. When webpage loading finishes, any unsent packets left in the timetable are simply dropped.

The choice of N_c and N_s depends on the amount of data budget we want to introduce to trade for privacy. We also find that it is better to set $1.5N_c < N_w < 3N_c$. W_{min} should be set according to the network bandwidth. W_{max} controls the width of the padding window. We set $W_{max} = 8$s in our experiments to cover all trace fronts while avoiding dropping too many dummy packets.

B. GLUE

1) Intuition: GLUE exploits another weakness of WF attacks — it is hard to decide the number of traces and further split them given ℓ consecutive web page loadings. Thus, GLUE inject "glue noise" **between** traces to connect them together. Then an attacker only see a long trace consisting of ℓ different visits. GLUE also integrates FRONT to protect the trace front.

2) Defense Design: GLUE switches in three modes, described as follows.

a) Front Mode: Starting in Front Mode, our defense waits for the client to visit a webpage. When the client does so, we will add dummy packets according to our FRONT defense, as described in previous sections. We will also sample the client's packet inter-arrival times here to obtain some distribution I. After the client finishes visiting the webpage, we sample $t_\Delta \in U(I_{\bar{20}}, I_{\bar{80}})$, wait for time t_Δ, then switch to Glue Mode.

b) Glue Mode: In Glue Mode, the client and proxy send each other dummy packets in such a way that it looks as if the client decided to visit a new, random webpage. They will do so for at most time d_{max}. They immediately stop doing so if the client actually decides to visit a webpage before d_{max} has passed: the client will notify the proxy to terminate Glue Mode as well. If the client dwells on the webpage for longer than d_{max}, the algorithm will consider the client inactive and return to Front Mode. Otherwise, it will go to Back Mode.

c) Back Mode: In Back Mode, the client is visiting another webpage. This is like Front Mode, with the difference that we add no dummy packets whatsoever. We still sample packet inter-arrival times and switch back to Glue Mode after waiting for a sampled t_Δ.

III. Preliminary Evaluation

We do experiments on Wang's dataset [2]. Their dataset contains 100 monitored webpages with 90 instances each together with 9000 unmonitored webpages. We use three state-of-art WF attacks [2]–[4] to test our defense. We compare our defenses with a state-of-art lightweight defense, WTF-PAD [5].

A. Evaluation of FRONT

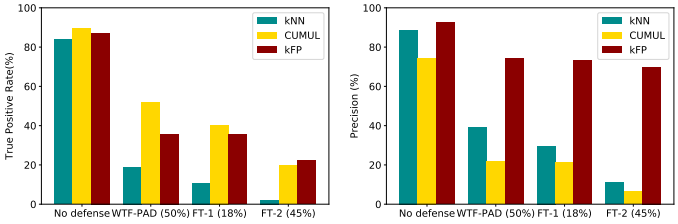

Fig. 1. FRONT performance against WF attacks. We also mark the data overhead on X-axis.

B. Evaluation of GLUE

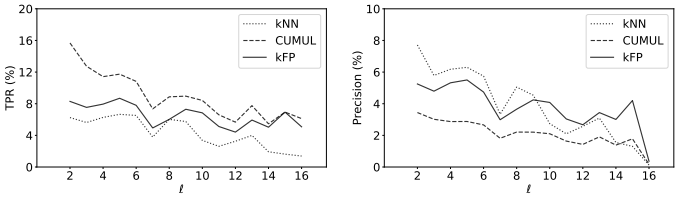

Fig. 2. GLUE performance vary ℓ against WF attacks.

IV. Declaration

This work was submitted to USENIX Security Symposium 2019 and is currently under review, thus unpublished.

References

[1] P. Syverson, R. Dingledine, and N. Mathewson, "Tor: The Second Generation Onion Router," in *USENIX Security Symposium*, 2004.

[2] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective Attacks and Provable Defenses for Website Fingerprinting," in *USENIX Security Symposium*, 2014.

[3] J. Hayes and G. Danezis, "k-fingerprinting: A Robust Scalable Website Fingerprinting Technique," in *USENIX Security Symposium*, 2016.

[4] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website Fingerprinting at Internet Scale," in *Network & Distributed System Security Symposium (NDSS)*, 2016.

[5] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright, "Toward an Efficient Website Fingerprinting Defense," in *European Symposium on Research in Computer Security*. Springer, 2016.