Huang and Du *Cybersecurity (2024)7:28*
https://doi.org/10.1186/s42400-024-00222-y

Cybersecurity

RESEARCH

Open Access

Break-Pad: effective padding machines for tor with break burst padding

Bin Huang[1*] and Yanhui Du[1]

Abstract

Website Fingerprinting (WF) attacks enable a local eavesdropper to use metadata of packet flow, such as size, timing, and direction, to infer the websites a user is visiting. This can damage the user privacy provided by anonymity systems such as Tor. Tor has implemented the WF defense called Circuit Padding Framework, which provides an interface for developers to implement their own defenses. However, these defenses in the framework were overcome by the Deep Fingerprinting (DF) attack. In this paper, we propose a novel defense approach called break burst padding (Break-Pad), which injects a random number of padding packets into an incoming burst once the number of consecutive incoming packets exceeds a set number. We integrated Break-Pad into the existing Circuit Padding Framework. In addition, we have implemented two padding machines named August and October in the new framework and conducted experiments to evaluate these machines. In the open-world setting, our results show that August, with 29% bandwidth overhead, reduces Tik-Tok's TPR by 14.48% and DF's TPR by 22%. October outperforms the best padding machine, RBB. With 36% bandwidth overhead, it drops Tik-Tok's TPR to 74.24% and DF's TPR to 65.36%. In the one-page setting, October further reduces the bandwidth overhead by 11% while achieving similar performance to RBB. In the information leak analysis, for the burst sequence feature of the traffic, October leaks at 2.453 bits, while the best comparable padding machine Interspace leaks at 2.629 bits.

Introduction

Tor (Dingledine et al. 2004) is one of the most popular anonymity systems to hide information about personal browsing websites. In Tor, a user establishes a multi-hop path (called a circuit) to visit a website and transmits fixed-size encrypted packets (called cells) on this path. So, any single node cannot link the user's identity with the website the user accessing. Unfortunately, Tor is vulnerable to a class of traffic analysis known as Website Fingerprinting (WF) (Wang et al. 2014; Kwon et al. 2015; Hayes and Danezis 2016; Panchenko et al. 2016; Rimmer et al. 2018; Sirinam et al. 2018; Bhat et al. 2019; Sirinam et al. 2019; Rahman et al. 2020; Se Eun et al. 2021; Cherubin et al. 2022; Shen et al. 2023; Deng et al. 2023). WF allows eavesdroppers to learn metadata of packet flow and feed the information into a machine learning classifier to train it. They then use the trained classifier to predict which pages the user visited even though the traffic is encrypted.

In response, many works have proposed defense algorithms to defeat WF attacks. Some defenses (Dyer et al. 2012; Cai et al. 2014a, b; Juarez et al. 2016; Lu et al. 2018; Abusnaina et al. 2020; Gong and Wang 2020; Al-Naami et al. 2021; Holland and Hopper 2022) morph the traffic according to predefined patterns, and some (Wang et al. 2014; Nithyanand et al. 2014; Wang and Goldberg 2017; Rahman et al. 2021; Gong et al. 2022) modify the traffic based on the reference traffic they generate, and some (De la Cadena et al. 2020; Henri et al. 2020) split the traffic and send it over multiple network paths. The Tor Project community is also concerned about such attacks and has deployed a WF defense, Circuit Padding Framework (padding spec 2019). The framework allows developers

*Correspondence:
Bin Huang
beenhuang@126.com
[1] People's Public Security University of China, No. 1, Muxidi Nanli, Xicheng District, Beijing, China

to implement their own defenses, which inject padding packets into time gaps in the packet flow without adding any latency to traffic. However, these defenses are either ineffective against state-of-the-art attacks, such as DF (Sirinam et al. 2018), or bring unacceptably high bandwidth overhead.

In this paper, we propose a novel defense called break burst padding (Break-Pad), which injects padding packets into incoming bursts to break burst patterns of traffic. Break-Pad utilizes padding based on the number of incoming packets: it sends padding packets once the number of incoming packets exceeds a threshold. The defense effectively disrupts burst patterns by splitting a long burst into a set of smaller-sized bursts. In addition, both padding packets and thresholds for Break-Pad are sampled from the probability distributions we set. The randomness of the number of padding packets and threshold, even for the same web page, makes the burst pattern of each trace different, which also helps to improve the effectiveness of our defense. And, we can limit the bandwidth overhead by adjusting the probability distribution of the defense.

We combined our defense with Circuit Padding Framework (padding spec 2019) and implemented two padding machines: August and October. August is a one-way padding machine with Break-Pad, which only allows the client machine to send padding packets. October is a two-way padding machine, which can inject padding packets in both directions.

To show the defense performance of our padding machines, we conducted experiments and compared them with other machines and WF defenses. Compared to other padding machines, in the open-world setting, August is effective against the best WF attacks: with 29% bandwidth overhead, it reduces Tik-Tok's (Rahman et al. 2020) TPR to 83.28% and DF's (Sirinam et al. 2018) TPR to 76.84%, and October outperforms the best padding machine, RBB (Mathews et al. 2018), but with less bandwidth overhead: it, with 11% less bandwidth overhead than RBB, reduces Tik-Tok's TPR to 74.24% while RBB only reduces it to 82.75%. To have a comprehensive evaluation of the machines, we also conduct the information leakage analysis (Li et al. 2018) and perform the evaluation in the one-page setting (Wang 2021). In the information leakage analysis, for the Burst category, October leaks at 2.453 bits, compared to the best machine Interspace (Pulls 2020) gets 2.629 bits. In the one-page setting, October limits the precision of the k-FP (Hayes and Danezis 2016) attack to 82.44%, while RBB achieved a precision of 82.35%. However, October incurs 11% less bandwidth overhead than RBB. Compared to other WF defenses, on various security mode datasets, August and October outperformed the best WF defense DFD (Abusnaina et al. 2020). In the larger open-world setting, both August and October significantly decrease the performance of the DF (Sirinam et al. 2018) attack as the unmonitored set increases.

We summarize the contributions of our work as follows:

- We propose a realistic and novel WF defense approach called break burst padding (Break-Pad) within the existing Circuit Padding Framework of the Tor network, which obfuscates traffic characteristics with a more efficient padding approach.
- We designed and implemented two padding machines with Break-Pad: August and October. Experimental results show that both machines have high defense performance with low bandwidth overhead in the open-world setting and the one-page setting.
- We utilized the distribution fitting approach to search the optimal parameters for the padding machines. To the best of our knowledge, we are the first to use this approach to fix the parameters.

We organize the rest of the paper as follows. We introduce the background and related work in section "Background and Related Work". We next give, In section "Preliminaries". In section "Break Burst Padding Defense", we proposed our defense, Break-Pad. We evaluate our padding machines in section "Evaluation" and give the discussion in section "Discussion". Finally, we conclude our work in section "Conclusion", and we share the source code for both machines in the "Appendix".

Background and related work

In this section, we summarize previous work on WF attacks and defenses. Then we describe the WF defense deployed on Tor, which is known as Circuit Padding Framework (padding spec 2019), and padding machines (Mathews et al. 2018; Pulls 2020; Kadianakis et al. 2021) proposed in the framework.

Website fingerprinting attacks

The WF attacks proposed early used machine learning models as classifiers with hand-crafted features as input (Wang et al. 2014; Kwon et al. 2015; Hayes and Danezis 2016; Panchenko et al. 2016). After that, a variety of WF attacks using deep learning were proposed (Rimmer et al. 2018; Sirinam et al. 2018; Bhat et al. 2019; Sirinam et al. 2019; Rahman et al. 2020; Se Eun et al. 2021; Cherubin et al. 2022; Shen et al. 2023; Deng et al. 2023).

1) *ML-based attacks:* The state-of-the-art ML-based WF attacks are Wang-kNN (Wang et al. 2014), k-FP (Hayes and Danezis 2016), and CUMUL (Panchenko

et al. 2016). Wang-kNN (Wang et al. 2014) extracts more than 3000 statistical features from the packet flow and feeds these features into the classifier based on the k-Nearest Neighbor algorithm (kNN) to predict the websites visited by users. k-FP (Hayes and Danezis 2016) also uses kNN as a classifier. However, unlike Wang-kNN, k-FP does not feed statistical features directly into the classifier, but rather a *website fingerprint* generated by a Generator as input. A *website fingerprint* is represented by a vector of leaf identifiers from a trained Random Forest. CUMUL (Panchenko et al. 2016) uses Support Vector Machine (SVM) as a classifier and takes the cumulative sum of packet lengths as input features to the classifier. These attacks are effective against undefended traffic. However, for defended traffic, the performance of these attacks will be significantly reduced.

2) *DL-based attacks:* The classic DL-based attacks are AWF (Rimmer et al. 2018), DF (Sirinam et al. 2018), Var-CNN (Bhat et al. 2019), Tik-Tok (Rahman et al. 2020). Rimmer et al. (2018) propose using a simple Convolution Neural Network (CNN) (Lecun et al. 1998) architecture in their Automatic Website Fingerprinting (AWF). In the closed-world setting (containing 900 classes), AWF obtained an accuracy of 91.79%. Sirinam et al. (2018) proposed the Deep Fingerprinting (DF) model, which utilizes a 1D-CNN architecture with more hidden layers. DF achieves over 98% accuracy on undefended traffic. And, DF gets more than 90% accuracy against traffic defended by the WTF-PAD (Juarez et al. 2016) defense. The results show that DF breaks WTF-PAD. Bhat et al. (2019) proposed Var-CNN, which integrates two classical ResNet-18 (He et al. 2016). One ResNet-18 model takes the direction sequence as input, and the other ResNet-18 model takes the timestamp sequence as input. Compared to DF, it gets better performance in the low-data setting. Rahman et al. (2020) proposed Tik-Tok, which utilizes the directional timing feature to effectively enhance the performance of the DF. The directional timing feature is a sequence of values, where each value is yielded by multiplying the packet's timestamp by its direction. In the closed-world setting, Tik-Tok achieved 97% accuracy on traffic defended by Walkie-Talkie (Wang and Goldberg 2017). The results show that Tik-Tok overcomes Walkie-Talkie.

Website fingerprinting defenses

To counter WF attacks, WF defenses (Panchenko et al. 2011; Dyer et al. 2012; Cai et al. 2014a, b; Nithyanand et al. 2014; Juarez et al. 2016; Cherubin et al. 2017; Wang and Goldberg 2017; Lu et al. 2018; Abusnaina et al. 2020; Gong and Wang 2020; De la Cadena et al. 2020; Henri et al. 2020; Al-Naami et al. 2021; Nasr et al. 2021; Rahman et al. 2021; Gong et al. 2022; Holland and Hopper 2022; Smith et al. 2022; Mathews et al. 2023) modify the original pattern of traffic to reduce the amount of information leaked to the adversary. Existing defenses typically inject, delay, merge, and split packets into the traffic or transmit the traffic over multiple network channels. We roughly categorize previous defenses into five broad categories: Fix-rate, Padding, Reference Trace, Traffic Splitting, and others.

(1) *Fix-rated Defenses:* These defenses aim to morph packet sequence patterns to appear similar or identical, preventing the adversary from distinguishing among them. BuFLO (Dyer et al. 2012) operates by sending fixed-sized packets at fixed intervals. If there is no real packet to be sent within the set time, it sends a dummy packet. CS-BuFLO (Cai et al. 2014a) is an improved version of BuFLO, which can adjust the transmission rate appropriately based on the network congestion. Tamaraw (Cai et al. 2014b) is proposed as a lightweight BuFLO that reduces defense overhead while guaranteeing defense effectiveness. RegulaTor (Holland and Hopper 2022) regularizes the *surges* in the packet sequences that often occur in download traffic. *surge* is defined as a large number of packets sent in a short period of time. While these defenses can counter WF attacks using simple algorithms and do not require additional resources (e.g., storage space) or packet sequence information from the target site, their bandwidth and latency overhead are extremely high, making them difficult to apply in practice.

(2) *Padding-based Defenses:* These defenses add dummy packets based on predefined rules to mask the original pattern of the traffic. WTF-PAD (Juarez et al. 2016) uses delay-based padding, where sending a padding packet is triggered when the chosen delay expires. It tries to obfuscate the packet interval time characteristics of the traffic. DFD (Abusnaina et al. 2020) only injects padding packets into the outgoing burst. After the client sends two consecutive packets, DFD starts injecting padding packets. The number of padding packets is half of the length of the previous outgoing burst. FRONT (Gong and Wang 2020) injects padding packets at the front of the traffic. It samples the total number of padding packets from the Uniform distribution, and samples the time of each padding packet in the Rayleigh distribution. It then sends the padding packets at the set times.

(3) *Traffic Splitting Defenses:* This class of defenses resists WF attacks by splitting traffic over multiple network channels so that none of the channels can get full information about the target website. The classical defenses are TrafficSilver (De la Cadena et al. 2020) and HyWF (Henri et al. 2020). TrafficSliver (De la Cadena et al. 2020) uses the circuit with multiple entry nodes, splits application traffic over multiple entry nodes for transmission, and merges traffic at the middle nodes.

HyWF (Henri et al. 2020) splits traffic over two different access points (e.g. one for Cellular network and the other for home WiFi) and merges traffic at a multipath-compatible Tor bridge.

(4) *Reference Trace-based Defenses:* These defenses morph the original traffic pattern of the web page from the reference trace they create. The two defenses, Glove (Nithyanand et al. 2014) and Supersequence (Wang et al. 2014), group web pages into clusters and create a *super-trace* for each cluster. When users visits a page in a cluster, it modifies the traffic pattern of the page by injecting and delaying packets according to the *super-trace* of that cluster. Walkie-Talkie (Wang and Goldberg 2017) has modified the browser so that it can communicate in half-duplex. And it molds the burst sequence of a sensitive page based on a non-sensitive page (which is a reference trace) to make the two pages look the same. Surakav (Gong et al. 2022) utilizes Generative Adversarial Networks (GAN) (Goodfellow et al. 2020) to generate sending patterns (which is a reference trace) and regulates target traffic based on the generated patterns.

(5) *Other Defenses:* Panchenko et al. (2011) proposed Decoy, which loads a decoy webpage simultaneously when a user visits a website. It mixes the traffic of two web pages together, making it hard for an attacker to get the "pure" traffic of a single web page. Recent research (Wang 2021) has shown that its bandwidth overhead is too high and does not guarantee the effectiveness of the defense.

Circuit padding framework and padding machines

Against the WF attacks, the Tor Project community has deployed the Circuit Padding Framework (padding spec 2019) based on the research results of Shmatikov and Wang (Shmatikov and Wang 2006) and Juarez et al. (2016). In this framework, a developer can implement his own padding defense by designing a padding machine, a finite state machine. In the machine, each state samples the inter-packet delay from a histogram or probability distribution and injects a padding packet if the chosen delay expires. They can define different histograms or probability distributions to perform different padding patterns.

Tor currently deploys two padding machines enabled by default: the Client-side Introduction Circuit Hiding machine (padding spec 2019) and the Client-side Rendezvous Circuit Hiding machine (padding spec 2019), both of which aim to hide features of the construction cell sequence of the client-side onion service circuit. Mathews et al. (2018) used histograms to implement two padding machines: Random Extend Bursts (REB) and Random Break Bursts (RBB). Both machines determine to send padding bursts with a 10% probability. When a padding burst is sent, they will decide whether to continue sending a padding packet with a 50% probability. Pulls (2020) used genetic programming to create the padding machine called Spring. Based on Spring, the author added the probabilistic state transitions approach to generate a new padding machine called Interspace. Kadianakis et al. (2021) proposed a padding machine called Preemptive Circuit Padding (PCP), which injects dummy onion handshakes in the preemptive phase.

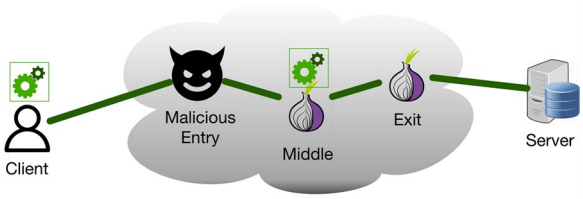


Fig. 1 The threat model, in which the client and the middle node participate in defense together against traffic analysis from the malicious entry node

Preliminaries

Threat model

Figure 1 illustrates the threat model. We assume that the attacker is a relay adversary, who is located on the entry node of the Tor circuit. As a node of the circuit, they are able to see cells (fixed-size packets) transferred on the circuit. Although he cannot see the contents of the cells, he can still obtain the type and direction of all cells transferred on the target circuit. Besides, the adversary is a passive observer who cannot insert, drop, modify, or delay packets.

For traffic analysis of the malicious entry relay, the client and the middle relay cooperate to defend against such attacks in Tor. In the Tor network, they only send padding cells to each other, and cannot delay cells. In addition, the middle relay automatically drops all padding cells from the client, so the exit relay and web server would not be affected.

Defense overhead

Following the methodology of prior works, we evaluate bandwidth and time overhead for the defenses. The bandwidth overhead is the ratio of the total number of dummy packets to the total number of real packets on the whole dataset. In our work, we further evaluate the incoming and outgoing bandwidth overhead. The incoming/outgoing bandwidth overhead is calculated as the total number of incoming/outgoing dummy packets divided by the total number of real incoming/outgoing packets. The time overhead is the ratio of the total extra time added by the defenses to the total time without defenses.

Metrics

We use multi-class classification metrics for the open-world setting to evaluate the defense performance. We first introduce TP-c, TP-i, FN, FP, and TN, and then describe how to calculate Precision, True Positive Rate (TPR), and False Positive Rate (FPR).

- True Positive with the correct website class (TP-c) is the number of monitored classes predicted as the monitored and the correct website class.
- True Positive with the incorrect website class (TP-i) is the number of monitored classes predicted as the monitored and the incorrect website class.
- False Negative (FN) is the number of monitoring classes predicted as unmonitored.
- False Positive (FP) is the number of unmonitored classes predicted as monitored.
- True Negative (TN) is the number of unmonitored classes predicted as unmonitored.

Precision is calculated as the number of monitored classes classified as monitored and correct website classes divided by the total number of predicted monitored classes.

$$Precision = \frac{TP\text{-}\,c}{TP\text{-}\,c + TP\text{-}\,i + FP} \tag{1}$$

TPR (Recall) is calculated as the number of monitored classes classified as monitored and correct website classes divided by the total number of monitored classes.

$$TPR = \frac{TP\text{-}\,c}{TP\text{-}\,c + TP\text{-}\,i + FN} \tag{2}$$

FPR is calculated as the number of unmonitored classes classified as monitored divided by the total number of unmonitored classes.

$$FPR = \frac{FP}{FP + TN} \tag{3}$$

Break burst padding defense

In this section, we introduce our break burst padding defense. We first discuss the motivation. Then we give an overview of our defense and describe how to fix the best parameters. Finally, we show two padding machines with break burst padding.

Motivation

The Circuit Padding Framework (padding spec 2019) uses delay-based padding, where sending a padding packet is triggered only when the chosen delay expires. However, it cannot meet the requirement of injecting padding packets in bursts with less bandwidth overhead. In traffic, generally, the inter-packet time of bursts is short and the inter-burst time is long. If we set a long delay, the defense cannot inject padding packets in the burst, so the traffic still retains the characteristics of the burst pattern. Since the burst pattern is not broken, making it successfully overcome by the DF (Sirinam et al. 2018) attack. If we set a short delay, the defense injects padding packets between almost every real packet or injects a large number of padding packets during the idle time of the traffic. It causes the defense to incur huge bandwidth overhead, e.g., the total number of padding packets is much larger than the total number of real packets. And, the delay-based padding is very difficult to set appropriate delays that allow the defense to inject sufficient padding packets in each burst and not to inject too many padding packets between bursts or during the idle time of the traffic. To deal with the above problems of the Circuit Padding Framework, we try to find a padding approach that can effectively break burst patterns of packet sequences in the existing Circuit Padding Framework with less defense overhead to resist WF attacks such as DF (Sirinam et al. 2018).

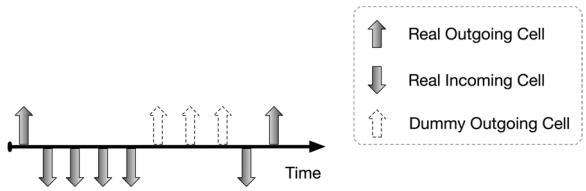


Fig. 2 Visualization of break burst for one time

Overview of break burst padding

We first introduce a type of padding approach called break burst. As shown in Fig. 2, a break burst means that it uses a padding burst B (containing 3 packets) to divide a real incoming burst R (containing 5 packets) into two short bursts R_1 (containing 4 packets) and R_2 (containing 1 packet), and B is located between them.

Break Burst Padding (Break-Pad) is designed by utilizing the break burst approach. Specifically, the padding position selection in Break-Pad is determined by the number of incoming packets: it sends some padding packets when the number of packets it receives exceeds the threshold. Moreover, rather than using a fixed padding pattern, the defense samples random numbers of thresholds and padding packets from the probability distributions each time, so that the same traffic will show different burst patterns. In addition, we set the probability distribution of our defense based on the data distribution of the traffic in the monitored class.

Algorithm 1 Break Burst Padding Algorithm

Input: Packet Sequence T
Output: Defended Packet Sequence
1: $n \leftarrow 0$
2: p samples from D_p, b samples from D_b
3: **for** each packet **in** T **do**
4: **if** packet is incoming **then**
5: $n \leftarrow n+1$
6: **if** $n = p$ **then**
7: Send b padding packets
8: $n \leftarrow 0$
9: p samples from D_p, b samples from D_b
10: **end if**
11: **end if**
12: **if** packet is outgoing and $n \neq 0$ **then**
13: $n \leftarrow 0$
14: p samples from D_p, b samples from D_b
15: **end if**
16: **end for**

Break-Pad works as shown in Algorithm 1. we first samples a threshold p and a padding burst b from D_p and D_b, respectively (Line 2). In each iteration, if the packet is incoming, the number of consecutive incoming packets n is added to 1. When n equals the threshold p, we successively send b padding packets and sample the new values of p and b (See Line 4–11). If the packet is outgoing and n is not equal to 0, we set n to its initial value of 0 and sample the new values of p and b (Line 12–15).

Break-Pad mainly destroys the burst sequence characteristics of the packet flow. We used Li's WeFDE (Li et al. 2018) tool to calculate the amount of information leaked by traffic defended by Break-Pad. WeFDE gets the amount of information that an attacker can obtain from F about W by calculating the mutual information $I(F; W)$, where W denotes the website information and F is a random variable representing the website fingerprint of W. The calculation formula is:

$$I(F; W) = H(W) - H(W|F) \tag{4}$$

Break-Pad reduces information leakage from 5.215 bits to 2.539 bits when only the client applies our defense, and to 2.453 bits when both the client and the relay apply our defense. In addition, we analyze in detail the information leakage of Break-Pad with respect to other categories of features in Section .

Setting parameters of probability distribution

Break-Pad is governed by four parameters in Table 1, where parameter p is a threshold that determines the number of incoming packets and parameter b determines the number of padding packets, and it gets random values

Table 1 Parameters for Break-Pad

Parameter	Description
p	Number of incoming packets
b	Number of padding packets
D_p	Probability distribution of p
D_b	Probability distribution of b

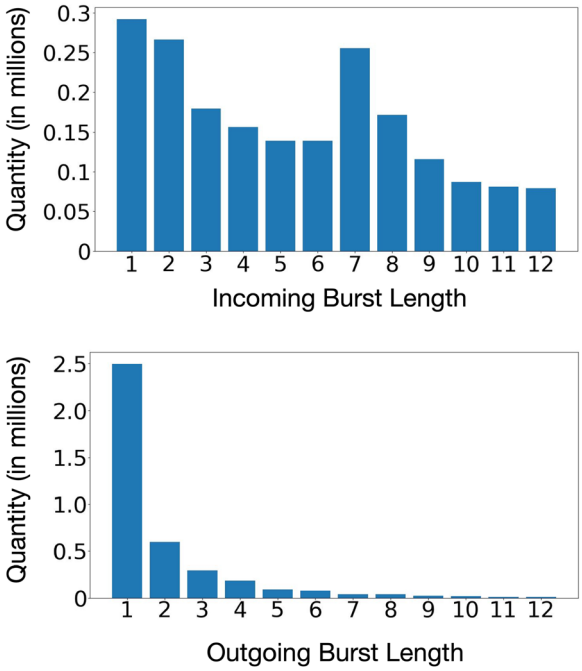


Fig. 3 Distribution of incoming and outgoing bursts on the Goodenough dataset

of p and b in each round by sampling from D_p and D_b, respectively. In this section, we introduce the approach for setting the parameters of the probability distributions D_p and D_b.

If the outgoing bursts (request packets) sent by a user in a dataset fit a probability distribution. We believe that the padding packets we sent should also match this distribution, otherwise, these padding packets will become another characteristic of the traffic. And, we believe that the number of incoming packets we set (padding points) should also match the distribution of incoming bursts in the dataset. Accordingly, we get the distribution parameters of the Break-Pad by fitting a probability distribution to the dataset.

The workflow of the parameter tuning approach is described as follows. We first count the number of incoming and outgoing bursts on the dataset and sort them by length. The results are shown in Fig. 3. Then, we select the top N bursts (e.g., when N is 10, we select all

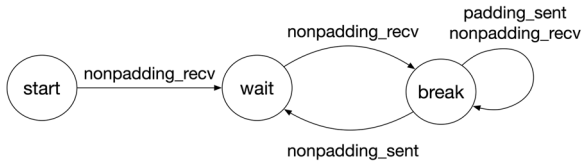


Fig. 4 State diagram for the client machine of August

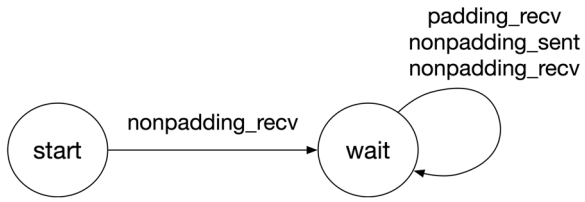


Fig. 5 State diagram for the relay machine of August

bursts with lengths less than or equal to 10) and fit the probability distribution to the data to obtain the parameters of the probability distribution. Using the above fitting approach, we choose different N bursts each time and fit the probability distribution to the data to obtain different parameters. Finally, we evaluate the performance of all parameters obtained by the fitting approach and pick the parameters that achieve the lowest TPR.

August: one-way Break-Pad machine

We designed the one-way padding machine with Break-Pad called August. The August machine consists of a client machine on the client and a relay machine on the middle node. We implemented the Break-Pad defense in the Break state of August's client machine.

In Fig. 4, the client machine has three states: Start, Wait, and Break. Start is the initial state of the machine. After receiving a padding negotiated cell sent by the relay machine, it transitions from Start to Wait. When the machine received a real cell sent by the server, it transitions to Break and samples the values of p (padding point) and b (padding burst) from D_p and D_b, respectively. In the Break state, the machine subtracts 1 from p each time it receives a cell. When p is 0, it continuously sends b padding cells. It then resamples to get the new values of p and b. If the machine sends a real cell and p is not 0, it transitions back to Wait.

Figure 5 shows all states of the relay machine, which has two states: Start and Wait. Same as the client machine, the initial state is still Start. It transitions from Start to Wait when receiving a client's padding negotiate cell. In Wait, it only drops the padding cells from the client. Since the relay machine is not allowed to send padding cells to the client, we do not need to set probability distributions for it.

Table 2 Parameters for August

Machine	Probability distribution	Parameter
Client	D_p^c	Type
		Param_1
		Param_2
	D_b^c	Type
		Param_1
		Param_2

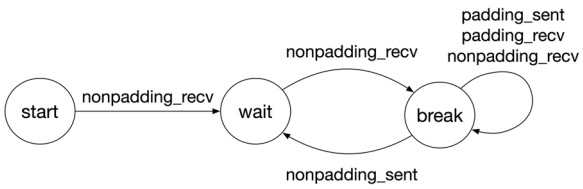


Fig. 6 State diagram for the relay machine of October

Table 2 summarizes all parameters of August we need to fix. D_p^c denotes the probability distribution of padding points of the client machine, and D_b^c denotes the probability distribution of padding bursts of the client machine. For each probability distribution, we need to set the type of the distribution, as well as two parameters for the distribution. We will fix the parameters using the approach introduced in Section .

October: two-way Break-Pad machine

In this subsection, we introduce the two-way padding machine with Break-Pad, October. Compared with August, October is able to send padding cells in both directions. So, we believe that October is more effective against WF attacks than August. We implemented the Break-Pad defense in the Break state of client and relay machines in October. Since the client machine of October is similar to the client machine of August, we will not introduce it again.

As shown in Fig. 6, the relay machine of October includes three states: Start, Wait, and Break. Start is the initial state. When a padding negotiate cell is received, it transitions to Wait. In Wait, it transitions to Break if receiving a real cell from the client. In the Break state, it runs the Break-Pad algorithm, which first samples the values of p and b. Then the value of p is subtracted by 1 every time a padding or real cell is received. When the value of p is 0, a padding burst including b cells is sent to the client. If it forwards a real cell sent from the server to the client and p is not 0, it transitions to Wait.

We show all the parameters for October in Table 3. D_p^c and D_b^c denote the probability distributions of padding

Table 3 Parameters for October

Machine	Probability distribution	Parameter
Client	D_p^c	Type
		Param_1
		Param_2
	D_b^c	Type
		Param_1
		Param_2
Relay	D_p^r	Type
		Param_1
		Param_2
	D_b^r	Type
		Param_1
		Param_2

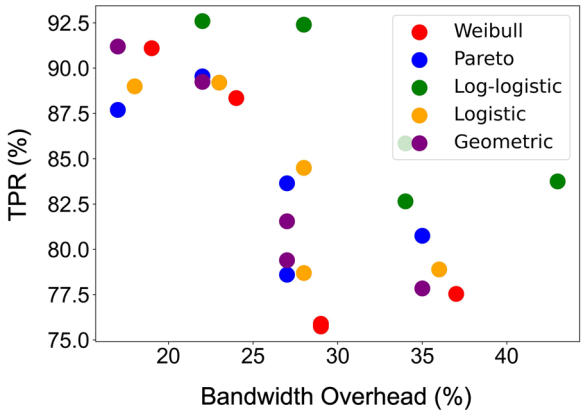


Fig. 7 Performance with different combinations in the August machine. Different distribution types of D_p are represented in different colors

points and padding bursts for the client machine, respectively, and D_p^r and D_b^r denote the probability distributions of padding points and padding bursts for the relay machine, respectively.

Evaluation

Here we evaluate August and October against other machines and the WF defenses. We first describe the dataset we used and the parameter tuning for both machines. Then we compare August and October with other machines and other WF defenses in several aspects.

Dataset

In this study, we use the Goodenough dataset[1], which is collected by Pulls and used to evaluate the performance of Spring and Interspace in his work (Pulls 2020). The Goodenough dataset contains three separate datasets that are based on the Tor Browser's three security settings: Standard, Safer, and Safest. The Standard is the default security level for the Tor Browser, allowing all content and scripts on a web page; The Safer disallows scripts from running on non-HTTPS sites and blocks some dynamic contents such as audio and video; The Safest disallows scripts from running on all websites and blocks more dynamic content from loading, including fonts, icons, math symbols, images, audio, and video.

In the Goodenough dataset, for each security mode, the dataset consists of 10,000 monitored instances from 50 websites (in which each of the 10 subpages per website was visited 20 times) and 10,000 unmonitored instances from 10,000 websites (in which each of the index pages of the website was visited 1 time).

[1] https://github.com/pylls/padding-machines-for-tor/tree/master/dataset

Parameter tuning

type of probability distribution

To find the best combination of distribution types for D_p and D_b, we tried all the distribution types provided by Tor in D_p and D_b of the August machine. We exclude the uniform distribution by observing the burst distribution of the Goodenough dataset (see Fig. 3 in Section). To reduce computational complexity and obtain results more quickly, we pick up incoming bursts with lengths less than or equal to 10 (the value of N_{in}^c is 10) and outgoing bursts with lengths less than or equal to 20 (the value of N_{out}^c is 20). Then, we fit the different types of distributions to the chosen burst data (incoming or outgoing bursts) to obtain the corresponding parameters for each type of distribution. And we set different distribution types and corresponding parameters in the D_p and D_b of the August machine to generate multiple machines containing different distributions. Finally, DF (Sirinam et al. 2018) was used to evaluate the defense performance of all machines. Figure 7 shows bandwidth overhead and DF's TPR for all machines. When D_p is set to the Weibull distribution and D_b is set to the Pareto distribution, this combination of distributions gets the lowest TPR (about 75%) with the 29% bandwidth overhead. Therefore, in the following experiments, we set D_p to the Weibull distribution and D_b to the Pareto distribution by default.

Parameters of probability distribution

The parameters can determine the location, shape, or scale of the distribution, so we get the best distribution by adjusting the parameters. And, we fit only the training data to obtain the parameters of the distribution.

Table 4 shows the search range and final values of N_{in}^c and N_{out}^c for the August machine. N_{in}^c represents incoming burst data with burst length less than or equal to the

Table 4 Search range and final values for August

Hyperparameter	Search range	Final
N_{in}^{c}	[5 ... 50]	10
N_{out}^{c}	[5 ... 50]	20

Table 5 Search range and final values for October

Hyperparameter	Search range	Final
N_{in}^{c}	[5 ... 50]	20
N_{out}^{c}	[5 ... 50]	25
N_{in}^{r}	[5 ... 50]	25
N_{out}^{r}	[1 ... 15]	3

value of N_{in}^{c}, and N_{out}^{c} represents outgoing burst data with burst length less than or equal to the value of N_{out}^{c}. We fit the default probability distributions to the N_{in}^{c} data and the N_{out}^{c} data to get the parameter values of D_{p}^{c} and D_{b}^{c} for August's client machine, respectively. When N_{in}^{c} is 10 and N_{out}^{c} is 20, the combination of parameters obtained by fitting achieves the lowest DF's TPR. The specific parameter values for August are shown in "Appendix A.1".

We present the search range and final values for the October machine in Table 5. N_{in}^{c} and N_{out}^{c} are incoming burst data and outgoing burst data, respectively, which are fitted to produce parameters for the client machine; N_{in}^{r} and N_{out}^{r} are incoming burst data and outgoing burst data, respectively, which are fitted to produce parameters for the relay machine. We find that the best results are yielded when N_{in}^{c} is 20, N_{out}^{c} is 25, N_{in}^{r} is 25, and N_{out}^{r} is 3. (see specific parameter values for the October machine in "Appendix B").

Performance in open-world

Experimental setting

Similar to prior experiments on defenses, we also conducted simulation experiments. We first implemented the Break-Pad defense and integrated it into the Circuit Padding Framework (padding spec 2019) on Tor 0.4.7.8. We then add the circuit padding simulator (Circpad-Sim[2]) developed by Pulls (Pulls 2020) to the modified Tor. Circpad-Sim uses Tor's unit test framework to simulate applying padding machines to generate defended traces. In addition, we rewrite the Python script for running the simulator inspired by Pulls's script. In this experiment, we only used the Standard dataset from Goodenough.

[2] https://github.com/pylls/circpad-sim

Table 6 Bandwidth overhead (BW) for padding machines (%)

Machine	Outgoing BW	Incoming BW	Total BW
Spring	324	100	123
Interspace	542	111	157
REB	47	48	48
RBB	421	5	47
August	**281**	**0**	**29**
October	**191**	**18**	**36**

Our work is shown in bold

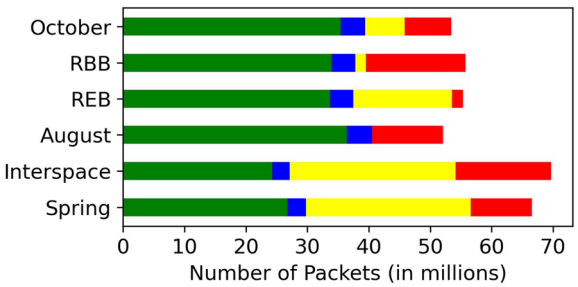


Fig. 8 Components of defended traces. A trace is composed of real incoming packets (green), real outgoing packets (blue), padding incoming packets (yellow), and padding outgoing packets (red)

We applied CUMUL (Panchenko et al. 2016), k-FP (Hayes and Danezis 2016), DF (Sirinam et al. 2018), and Tik-Tok (Rahman et al. 2020) as benchmarks to evaluate the effectiveness of the machines. CUMUL based on SVM is heavily dependent on the correct parameters, so we performed parameter tuning on the candidate parameters suggested by Panchenko et al. (2016) and found the optimal parameters by grid search with cross-validation. All classifiers extracted features from the first 5000 cells of the defended trace.

Defense overhead

The bandwidth overheads for each machine are presented in Table 6. The bandwidth overhead of Spring and Interspace is over 123%, which indicates that the padding packets are more than the real ones. From Fig. 8, both Spring and Interspace add too many incoming padding packets, resulting in high bandwidth overhead. By contrast, in our design, August does not generate incoming padding packets, so the bandwidth overhead is less than 30%.

In Table 6, The bandwidth overhead of REB and RBB is less than 50%. However, REB injects much more incoming padding packets than outgoing padding packets from Fig. 8. On the contrary, the outgoing padding packets of RBB are more than the incoming padding packets. We believe that too many incoming/outgoing padding packets than the other padding packets will not obtain good performance. In

Table 7 Defense performance for padding machines in open-world (%)

Machine	Precision				TPR				FPR				BW
	CUMUL	k-FP	DF	Tik-Tok	CUMUL	k-FP	DF	Tik-Tok	CUMUL	k-FP	DF	Tik-Tok	
None	92.90	95.94	95.48	95.04	94.2	91.26	98.4	97.76	6.55	2.78	4.12	4.39	0
Spring	17.06	44.32	39.70	46.15	13.97	10.02	34.76	45.89	35.16	3.07	29.62	26.13	123
Interspace	8.55	32.19	31.13	36.13	6.02	5.36	16.16	31.29	33.2	2.59	17.58	23.54	156
REB	78.62	83.89	87.52	91.39	80.32	59.46	93.09	92.57	15.83	5.29	10.75	6.6	48
RBB	31.86	60.88	64.51	82.47	28.16	22.03	71.63	82.75	33.93	5.10	27.69	12.29	47
August	**48.06**	**68.47**	**66.76**	**86.83**	**46.38**	**30.45**	**76.84**	**78.15**	**30.68**	**5.38**	**27.15**	**6.5**	**29**
October	**38.42**	**63.24**	**64.98**	**78.73**	**36.88**	**23.20**	**65.36**	**69.25**	**35.29**	**4.78**	**22.34**	**9.65**	**36**

Our work is highlighted in bold

comparison, for October, the incoming padding packets are similar to the outgoing padding packets.

Results

We show the results in Table 7. When there is no machine applied, CUMUL (Panchenko et al. 2016), k-FP (Hayes and Danezis 2016), DF (Sirinam et al. 2018), and Tik-Tok (Rahman et al. 2020) achieve more than 92% Precision, more than 91% TPR, and less than 6.5% FPR. DF has the highest TPR (98.4%) and k-FP has the highest Precision (95.94%) and the lowest FPR (2.78%) among all attacks. The results show that WF attacks are highly effective in the open-world setting when no machine is used. REB has the worst performance among all machines. With 48% bandwidth overhead, REB only reduced Tik-Tok's Precision by 3.65% (95.04% → 91.39%), reduced Tik-Tok's TPR by 5.19% (97.76% → 92.57%), and increased Tik-Tok's FPR by 2.21% (4.39% → 6.6%). Spring is more effective against all attacks. The Precision of Tik-Tok is reduced to 46.15%, the TPR of Tik-Tok is reduced to 45.89%, and the FPR of CUMUL is increased to 35.16%, however, it brings a massive bandwidth overhead (123%). By contrast, August, only with 29% bandwidth overhead, dropped the Precision of Tik-Tok from 95.04% to 86.83%, dropped the TPR of Tik-Tok from 97.76% to 78.15%, and increased the FPR of CUMUL from 6.55% to 30.68%.

Interspace has the best defense performance against all attacks. It achieved the lowest Tik-Tok Precision (36.13%) and the lowest Tik-Tok TPR (31.29%) in all results. However, Interspace also requires the highest bandwidth overhead (156%) among all machines. RBB achieves good defensive performance against all attacks. RBB reduced Tik-Tok's Precision from 95.04% to 82.47%, reduced Tik-Tok's TPR from 97.76% to 82.75%, and increased CUMUL's FPR from 6.55% to 33.93%, with 47% bandwidth overhead. Compared with RBB, October, with less bandwidth overhead, gets better defense performance. With 11% less bandwidth overhead compared to RBB, it further decreased Tik-Tok's Precision by 3.74% (82.47% → 78.73%), decreased Tik-Tok's TPR by 13.5% (82.75% → 69.25%), and increased CUMUL's FPR by 1.36% (33.93% → 35.29%).

Information leakage analysis

To gain further insight into the machines, we performed information leakage analysis using Website Fingerprint Density Estimation (WeFDE) proposed by Li et al. (2018). WeFDE calculates the mutual information of a website W and a fingerprint F of W to estimate the amount of information that an adversary learns from F about W. It consists of two components: Website Fingerprint Modeler and Mutual Information Analyzer. Website Fingerprint Modeler uses Adaptive Kernel Density Estimate (AKDE) (Rosenblatt 1956) to model the probability density function of features and produce an estimate of the information leakage for each feature. Mutual Information Analyzer is used for feature dimension reduction. It only picks out the top 100 most informative non-redundant features and clusters the features. The redundant features are those that have higher mutual information than a threshold value (0.9).

Experimental setting

we evaluate the information leakage for each feature category for machines. Following their methodology, we compute the information leakage for 14 feature categories used in Li et at. (2018). Since some categories contain a large number of features, we prune the redundant features with a threshold of 0.9 and pick the top non-redundant 50 features in these categories (Transposition, Interval-I, Interval-II, Interval-III, and Pkt. Distribution). In calculating the amount of information leakage for each category, we did not cluster features within the category.

Results

Table 8 and Fig. 9 shows the results. All machines greatly reduce leakage of certain types (Burst, First 20, First 30,

Table 8 Information Leakage by feature category (bits) Coloring: (x > 5.0) red, (4.0 < x < 5.0) orange, (3.0 < x < 4.0) yellow, (x < 3.0) uncolored

Feature Category	None	Spring	Interspace	REB	RBB	August	October
Pkt. Count	4.752	3.797	3.885	4.495	3.559	3.873	3.632
Time	4.709	2.343	2.462	3.317	3.276	3.363	3.287
Ngram	5.643	5.643	5.496	5.643	5.641	5.643	5.642
Transposition	5.643	4.084	4.174	5.589	3.500	4.389	4.009
Interval-I	5.643	5.549	5.635	5.642	5.643	5.643	5.643
Interval-II	5.643	5.643	5.627	5.643	5.639	5.643	5.643
Interval-III	5.643	5.641	5.637	5.643	5.640	5.643	5.643
Pkt. Distribution	5.643	5.637	5.595	5.643	5.638	5.642	5.639
Burst	5.215	2.367	2.629	3.823	2.566	2.539	2.453
First 20	5.100	1.169	1.470	0.881	1.021	1.760	0.977
First 30	1.486	0.085	0.049	0.236	0.110	0.208	0.187
Last 30	0.897	0.288	0.204	0.452	0.135	0.206	0.158
Pkt. per Second	5.643	5.612	5.630	5.643	5.643	5.643	5.643
CUMUL	5.583	4.948	3.141	5.614	5.376	5.494	5.424

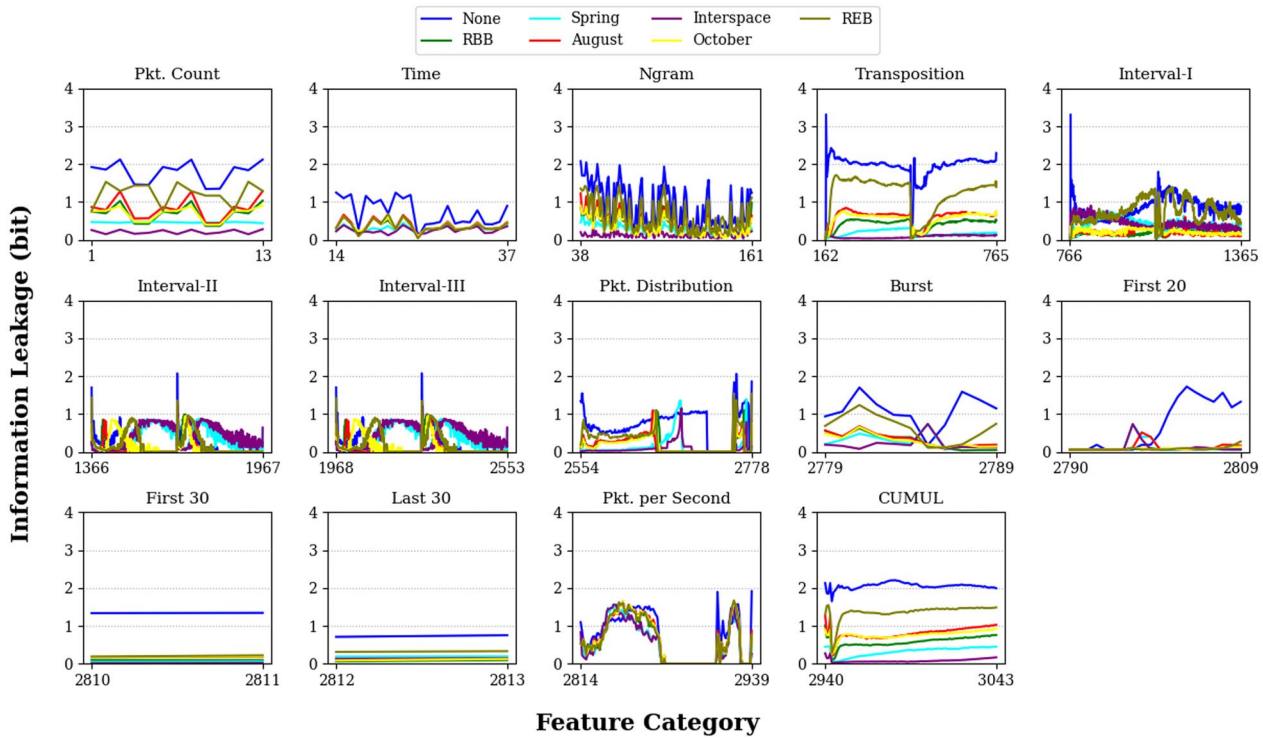


Fig. 9 Information Leakage by feature category

and Last 30). For the Burst category, October leaks at 2.453 bits which is the lowest leakage except Spring's. It means that October is effective to break the burst pattern with low bandwidth overhead. Some categories (Pkt. Count, Time, and Transposition) of information leakage have also decreased. We found that some categories (Ngram, Interval-I, Interval-II, Interval-III, Pkt. Distribution, and Pkt. per Second) still produce high information leakage. Compared with the very low TPR of CUMUL (Panchenko et al. 2016), the CUMUL category in the August and the October machines have high information leakage, 5.494 bits and 5.424 bits. However, it can be shown in Fig. 9 that August and October significantly reduce the information leakage

Table 9 Defense performance of padding machines against k-FP attack in the one-page setting (%)

Machine	Precision	TPR	FPR	Bandwidth
None	96.02	98	4.05	0
Spring	75.47	79.79	26.46	123
Interspace	70.32	77.25	33.06	156
REB	92.58	96.33	7.99	48
RBB	82.35	89.18	19.64	47
August	**86.80**	**91.95**	**14.25**	**29**
October	**82.44**	**89.30**	**19.40**	**36**

Our work is marked in bold

Performance in one-page

Wang (2021) proposes a higher evaluation standard (called one-page setting) to analyze WF defense performance, where there is only one monitored class and one unmonitored class. The monitored class contains only some instances of the same webpage that the attacker wants to identify.

Experimental setting

To show the defense effect of our padding machines, we performed the same experiments in the one-page setting. Each time, we randomly selected 200 instances from one website of the monitored class as positive samples and 200 instances from the unmonitored class as negative samples and performed binary classification with k-FP (Hayes and Danezis 2016). We repeated this evaluation 50 times each time using a different website of the monitored class (the monitored class of the Standard dataset contains 50 websites) and calculated the average of Precision, TPR, and FPR for all results as the final result of the experiment.

Results

Table 9 summarizes the results. On the undefended dataset, k-FP got 96.02% Precision, 98% TPR, and 4.05% FPR, showing that it is highly effective in identifying single webpages. Although Spring and Interspace get the best defensive performance among all machines, they carry too much bandwidth overhead. Compared to August, Spring only reduced Precision by 11.33% (86.80% → 75.47%), TPR by 12.16% (91.95% → 79.79%), and increased FPR by 12.21% (14.25% → 26.46%), but increased bandwidth overhead by 94% (29% → 123%). Compared with Spring, with 33% more bandwidth overhead, Interspace only decreased TPR by 2.54% (79.79% → 77.25%). REB is the least effective defense. With 48% bandwidth overhead, it only reduced Precision by 3.44% (96.02% → 92.58%), TPR by 1.67% (98% → 96.33%), and increased FPR by 3.94% (4.05% → 7.99%). RBB performed slightly better than August by further reducing Precision by 4.45% (86.80% → 82.35%), reducing TPR by 2.77% (91.95% → 89.18%), and increasing FPR by 5.39% (14.25% → 19.64%). However, RBB incurs 18% more bandwidth overhead than August. October gets good defense performance with less bandwidth overhead. While achieving similar performance to RBB, October further reduced bandwidth overhead by 11% (47% → 36%).

Performance against other WF defenses

We evaluated the performance of these two machines (August and October), as well as other WF defenses, using datasets based on three security modes in the Tor Browser Bundle (TBB): Standard, Safer, and Safest.

Experimental setting

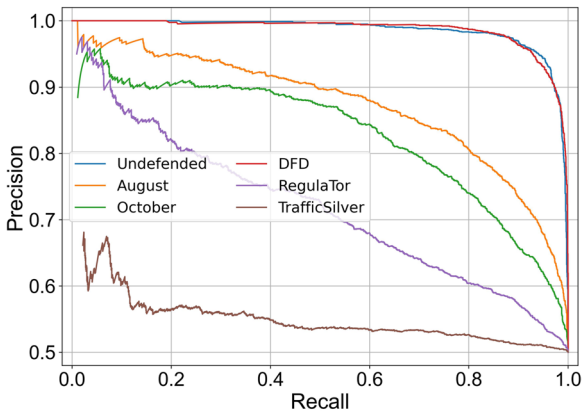
Based on the original paper, we have re-implemented the WF defenses, including DFD (Abusnaina et al. 2020), RegulaTor (Holland and Hopper 2022), and TrafficSilver (De la Cadena et al. 2020). There are two versions of RegulaTor: RegulaTor-Light and RegulaTor-Heavy. For this experiment, RegulaTor-Light, which has less defense overhead, is chosen. TrafficSilver adopts 3 circuits and the batch-weighted random (BWR) splitting strategy. We evaluated the effectiveness of the defenses using Tik-Tok (Rahman et al. 2020). To get more features of the traffic, Tik-Tok extracted features from the first 10,000 packets of traffic rather than the 5,000 packets in the original paper.

Defense overhead

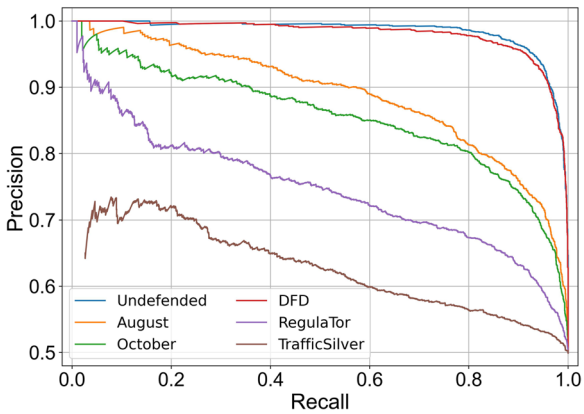
Table 10 shows the defense overhead for each defense on the various security model datasets. In all three datasets, DFD always brings about 55% bandwidth overhead. On Standard and Safer datasets, RegulaTor brings 24% bandwidth overhead and around 33% time overhead. On the Safest dataset, the bandwidth overhead of RegulaTor increased to 56% due to the reduction in actual traffic. TrafficSilver uses only traffic splitting, so its bandwidth and time overhead are both 0%. For various TBB security modes, August and October always incur lower bandwidth overheads of 29% and 36%, respectively. August and October in the Circuit Padding Framework do not delay any packets, so they both have 0% time overhead. We have found that both machines are padding based on actual traffic, and therefore have been maintaining a stable bandwidth overhead.

Results

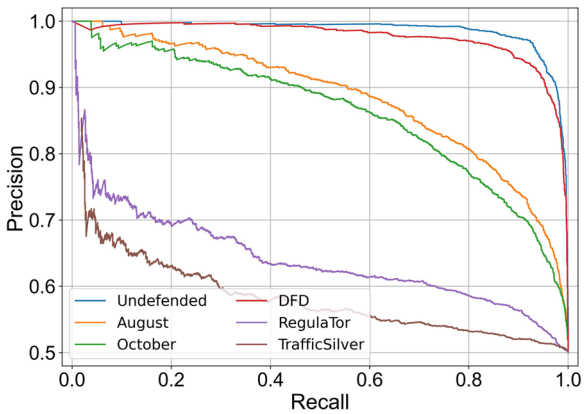
Figure 10 shows the precision-recall curves for Tik-Tok (Rahman et al. 2020) against the WF defenses on the dataset of the three security modes. As expected, Tik-Tok achieved high precision and recall in the three undefended datasets, showing that the attack is highly effective. DFD has the worst performance among all



(a) Standard



(b) Safer



(c) sfaest

Fig. 10 Precision-recall curves of Tik-Tok against WF defenses in Open-world

defenses. On both Standard and Safer datasets, it did almost nothing to reduce Tik-Tok's classification results. RegulaTor effectively reduces the performance of Tik-Tok. Its performance is better than the two padding machines in the Circuit Padding Framework and worse than TrafficSilver. However, RegulaTor imposes the time overhead of about 35%, which is why it cannot be deployed on the Tor network. At present low-latency anonymity systems such as Tor prohibit WF defenses from using delay packet operations. TrafficSilver gets the best defense performance. Particularly on Standard dataset, TrafficSilver kept Tik-Tok's precision rate less than 70%. Tik-Tok's classification effectiveness is significantly reduced by the fact that the adversary is only able to obtain approximately one-third of the traffic. However, the implementation of TrafficSilver in the Tor network requires modification of the existing Tor protocol. Compared to other WF defenses, August and October significantly decrease the performance of Tik-Tok on datasets with various security modes. Moreover, August and October do not utilize delayed packet operations and do not need to modify the Tor protocol, so they can be deployed directly into the Tor network.

Performance in larger open-world

In this subsection, we evaluate the performance of both padding machines in the larger open-world setting with significantly more negative than positive samples. As far as we know, the largest unmonitored dataset contains the index pages of 400,000 websites collected by Rimmer et al. (AWF400K)[3].

Experimental setting

Using the AWF400K dataset, we evaluate the defense performance against different sizes of the unmonitored set (10K, 50K, 100K, 200K, and 400K). Since AWF400K does not include packet timing, we choose the DF (Sirinam et al. 2018) attack that does not adopt packet timing features. The positive samples are from 5 randomly selected websites in the Goodenough dataset, totaling 1,000 instances.

Results

Figure 11 shows the performance of both August and October in the larger open-world setting. The results show that the performance of both defenses improves significantly with increasing size of the unmonitored set. For the 10K unmonitored websites, August and October reduced the DF's precision to 75.58% and 73.87%, respectively. Against the moderate-sized unmonitored websites of 100K size, August and October further dropped the DF's precision to 60.55% and 58.29%, respectively. For the largest unmonitored websites of 400K size, August and October further reduced the DF's precision to 40.60% and 39.58% respectively. Thus, we intuitively believe that

[3] https://github.com/DistriNet/DLWF

Table 10 Bandwidth (BW) and Time overhead for WF defenses for Standard, Safer, and Safest TBB Security Modes (%)

Defense	Standard		Safer		Safest	
	BW	Time	BW	Time	BW	Time
DFD	55	0	55	0	54	0
RegulaTor	24	35	24	33	56	35
TrafficSilver	0	0	0	0	0	0
August	**29**	**0**	**29**	**0**	**29**	**0**
October	**36**	**0**	**35**	**0**	**35**	**0**

Our work is shown in bold

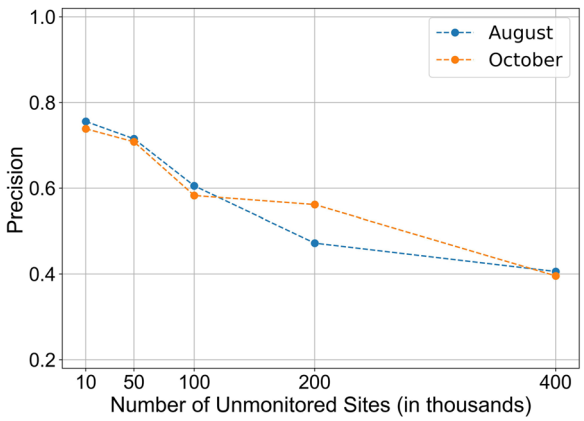


Fig. 11 Performance for padding machines with growing numbers of unmonitored sites

both defenses would perform more well in the real world, where there are 200 million active websites[4].

Discussion

Padding based on delay or based on the number of incoming packets? In the padding based on delay, the expiration of the set delay triggers the injection of padding packets. So, the defense tends to send more padding packets than intended in the case of network congestion. In reality, congestion in the underlying network or relays in Tor often causes the latency of incoming packets. And these latencies lead the defense to send extra padding packets. The extra padding packets not only increase bandwidth overhead but also make the network more congested.

In contrast, padding based on the number of incoming packets injects padding packets only when the number of incoming packets exceeds the set number. So, even if the incoming packets are delayed, the defense does not send additional padding packets.

Should we evaluate the impact of defense on the Tor network? To evaluate the effectiveness of the defense, prior works either conduct trace simulations, in which the simulator adds or delays packets according to the defense scheme; or the proposed defense is implemented as a pluggable transport (bridges and pluggable transports 2016), which obfuscates the traffic between a single client and a node in the live Tor network. And a defense that achieves worse attack results than other defenses is considered a better defense. However, these studies ignore the impact on the Tor network when all Tor users have deployed the proposed defense. The study of Wither et al. (2022) shows that if the padding-based defense were deployed on a large scale, the added padding packets will increase latency in the Tor network. This is a bad situation for Tor. Therefore, for a more comprehensive evaluation of defense, we should evaluate the impact on the Tor network after the proposed defense is deployed on a large scale. However, we are currently unable to deploy the WF defense on a large scale in the live Tor network, nor have we found a suitable simulator for testing.

Conclusion

In this paper, we presented Break-Pad, a practical defense that can be effective against the best WF attacks. Break-Pad decides whether to send padding packets based on the number of consecutive incoming packets. It selects random numbers of thresholds and padding packets each round, which are sampled from predefined probability distributions. In addition, we integrated our defense into Circuit Padding Framework in the Tor source code. In the modified framework, we implemented two padding machines, August and October. August is a one-way padding, lightweight defense where only the client machine sends padding packets using the Break-Pad approach. October is a two-way padding defense, where both the client and the relay machines can send padding packets to each other.

To show the effectiveness of our padding machines, we compared our padding machines with the previous best machines extensively in both the open-world setting and the one-page setting. Our open-world results show that August reduced Tik-Tok's TPR by 19.61% with only 29% bandwidth overhead. October outperformed RBB against the state-of-the-art attacks. With 11% less bandwidth overhead than RBB, October further reduced Tik-Tok's TPR by 13.5%. In the one-page setting, at a similar k-FP's

[4] https://websitesetup.org/news/how-many-websites-are-there/

Precision and TPR as RBB, the bandwidth overhead of October was further reduced by 11%. Additionally, we evaluate the information leakage for the traces defended by the machines. In the information leakage analysis, for the Burst category, October leaks 2.453 bits while the best machine Interspace leaks 2.629 bits. Compared to other WF defenses, August and October outperform the state-of-the-art WF defense DFD on all three security mode datasets. In the larger open-world setting, both August and October consistently reduced the DF's performance with increasing unmonitored sets.

Appendix A: Source code for August

Appendix A.1: Client machine

```
circpad_machine_spec_t *client_machine = tor_malloc_zero(sizeof(circpad_machine_spec_t));
client_machine->name = "august_client";
client_machine->is_origin_side = 1;
client_machine->target_hopnum = 2;
client_machine->conditions.apply_state_mask = CIRCPAD_CIRC_STREAMS;
client_machine->conditions.apply_purpose_mask = CIRCPAD_PURPOSE_ALL;
client_machine->allowed_padding_count = 1500;
client_machine->max_padding_percent = 50;

circpad_machine_states_init(client_machine, 3);
client_machine->states[0].next_state[CIRCPAD_EVENT_NONPADDING_SENT] = 1;
client_machine->states[1].next_state[CIRCPAD_EVENT_NONPADDING_RECV] = 2;

client_machine->states[2].contin_recv_length_dist.type = CIRCPAD_DIST_WEIBULL;
client_machine->states[2].contin_recv_length_dist.param1 = 1.6654098669893893;
client_machine->states[2].contin_recv_length_dist.param2 = 5.636855668373919;

client_machine->states[2].contin_padding_sent_length_dist.type =  CIRCPAD_DIST_PARETO;
client_machine->states[2].contin_padding_sent_length_dist.param1 = 1.979057415032052;
client_machine->states[2].contin_padding_sent_length_dist.param2 = 0.03733630342223472;

client_machine->states[2].contin_includes_padding_recv = 1;
client_machine->states[2].next_state[CIRCPAD_EVENT_PADDING_SENT] = 2;
client_machine->states[2].next_state[CIRCPAD_EVENT_NONPADDING_RECV] = 2;
client_machine->states[2].next_state[CIRCPAD_EVENT_NONPADDING_SENT] = 1;

client_machine->machine_num = smartlist_len(origin_padding_machines);
circpad_register_padding_machine(client_machine, origin_padding_machines);
```

Appendix A.2: Relay machine

```
circpad_machine_spec_t *relay_machine = tor_malloc_zero(sizeof(circpad_machine_spec_t));
relay_machine->name = "august_relay";
relay_machine->is_origin_side = 0;
relay_machine->allowed_padding_count = 1500;
relay_machine->max_padding_percent = 50;

circpad_machine_states_init(relay_machine, 2);
relay_machine->states[0].next_state[CIRCPAD_EVENT_NONPADDING_RECV] = 1;
relay_machine->states[1].next_state[CIRCPAD_EVENT_NONPADDING_SENT] = 1;
relay_machine->states[1].next_state[CIRCPAD_EVENT_PADDING_RECV] = 1;
relay_machine->states[1].next_state[CIRCPAD_EVENT_NONPADDING_RECV] = 1;

relay_machine->machine_num = smartlist_len(relay_padding_machines);
circpad_register_padding_machine(relay_machine, relay_padding_machines);
```

Appendix B: Source code for October

Appendix B.1: Client machine

```
circpad_machine_spec_t *client_machine = tor_malloc_zero(sizeof(circpad_machine_spec_t));
client_machine->name = "october_client";
client_machine->is_origin_side = 1;
client_machine->target_hopnum = 2;
client_machine->conditions.apply_state_mask = CIRCPAD_CIRC_STREAMS;
client_machine->conditions.apply_purpose_mask = CIRCPAD_PURPOSE_ALL;
client_machine->allowed_padding_count = 1500;
client_machine->max_padding_percent = 50;

circpad_machine_states_init(client_machine, 3);
client_machine->states[0].next_state[CIRCPAD_EVENT_NONPADDING_SENT] = 1;
client_machine->states[1].next_state[CIRCPAD_EVENT_NONPADDING_RECV] = 2;

client_machine->states[2].contin_recv_length_dist.type = CIRCPAD_DIST_WEIBULL;
client_machine->states[2].contin_recv_length_dist.param1 = 1.3832926042292748;
client_machine->states[2].contin_recv_length_dist.param2 = 8.766888541863576;

client_machine->states[2].contin_padding_sent_length_dist.type =  CIRCPAD_DIST_PARETO;
client_machine->states[2].contin_padding_sent_length_dist.param1 = 1.9667283364576538;
client_machine->states[2].contin_padding_sent_length_dist.param2 = 0.05282296143414936;

client_machine->states[2].contin_includes_padding_recv = 1;
client_machine->states[2].next_state[CIRCPAD_EVENT_PADDING_SENT] = 2;
client_machine->states[2].next_state[CIRCPAD_EVENT_NONPADDING_RECV] = 2;
client_machine->states[2].next_state[CIRCPAD_EVENT_PADDING_RECV] = 2;
client_machine->states[2].next_state[CIRCPAD_EVENT_NONPADDING_SENT] = 1;

client_machine->machine_num = smartlist_len(origin_padding_machines);
circpad_register_padding_machine(client_machine, origin_padding_machines);
```

Appendix B.2: Relay machine

```
circpad_machine_spec_t *relay_machine = tor_malloc_zero(sizeof(circpad_machine_spec_t));
relay_machine->name = "october_relay";
relay_machine->is_origin_side = 0;
relay_machine->allowed_padding_count = 1500;
relay_machine->max_padding_percent = 50;

circpad_machine_states_init(relay_machine, 3);
relay_machine->states[0].next_state[CIRCPAD_EVENT_NONPADDING_RECV] = 1;
relay_machine->states[1].next_state[CIRCPAD_EVENT_NONPADDING_RECV] = 2;

relay_machine->states[2].contin_recv_length_dist.type = CIRCPAD_DIST_WEIBULL;
relay_machine->states[2].contin_recv_length_dist.param1 = 1.1939537311219628;
relay_machine->states[2].contin_recv_length_dist.param2 = 2.2388583813756533;

relay_machine->states[2].contin_padding_sent_length_dist.type = CIRCPAD_DIST_PARETO;
relay_machine->states[2].contin_padding_sent_length_dist.param1 = 7.009539453953314;
relay_machine->states[2].contin_padding_sent_length_dist.param2 = -1.7523848634883286;

relay_machine->states[2].contin_includes_padding_recv = 1;
relay_machine->states[2].next_state[CIRCPAD_EVENT_PADDING_SENT] = 2;
relay_machine->states[2].next_state[CIRCPAD_EVENT_NONPADDING_RECV] = 2;
relay_machine->states[2].next_state[CIRCPAD_EVENT_PADDING_RECV] = 2;
relay_machine->states[2].next_state[CIRCPAD_EVENT_NONPADDING_SENT] = 1;

relay_machine->machine_num = smartlist_len(relay_padding_machines);
circpad_register_padding_machine(relay_machine, relay_padding_machines);
```

Acknowledgements
We would like to thank the anonymous reviewers for their helpful feedback.

Author contributions
BH: Conceptualization, Methodology, Software, Writing Manuscript. YD: Supervision

Funding
Not applicable.

Availability of data and materials
We publish all code used in this paper at https://github.com/beenhuang/padding-machine.

Declarations

Ethical approval and consent to participate
This article does not contain any studies with human participants or animals performed by any of the authors.

Competing interest
The authors declare that they have no competing interest in the publication of this study.

Received: 10 August 2023 Accepted: 20 February 2024
Published online: 01 October 2024

References

Abusnaina A, Jang RHO, Khormali A et al (2020) DFD: adversarial learning-based approach to defend against website fingerprinting. In: 39th IEEE conference on computer communications (IEEE INFOCOM), IEEE INFOCOM, pp 2459–2468. https://doi.org/10.1109/INFOCOM41043.2020.9155465

Al-Naami K, El-Ghamry A, Islam MS et al (2021) BiMorphing: a bi-directional bursting defense against website fingerprinting attacks. IEEE Trans Dependable Secure Comput 18(2):505–517. https://doi.org/10.1109/tdsc.2019.2907240

Bhat S, Lu D, Kwon A et al (2019) Var-CNN: a data-efficient website fingerprinting attack based on deep learning. In: Proceedings on privacy enhancing technologies, pp 292–310. https://doi.org/10.2478/popets-2019-0070

bridges, pluggable transports (2016) Bridges and pluggable transports. https://blog.torproject.org/tor-heart-bridges-and-pluggable-transports/

De la Cadena W, Mitseva A, Hiller J et al (2020) TrafficSliver: fighting website fingerprinting attacks with traffic splitting. In: ACM SIGSAC conference on computer and communications security (ACM CCS), pp 1971–1985. https://doi.org/10.1145/3372297.3423351

Cai X, Nithyanand R, Johnson R (2014a) Cs-buflo: A congestion sensitive website fingerprinting defense. In: Proceedings of the 13th workshop on privacy in the electronic society. Association for Computing Machinery, New York, pp 121–130. https://doi.org/10.1145/2665943.2665949

Cai X, Nithyanand R, Wang T et al (2014b) A systematic approach to developing and evaluating website fingerprinting defenses. In: 21st ACM conference on computer and communications security (CCS), pp 227–238. https://doi.org/10.1145/2660267.2660362

Cherubin G, Hayes J, Juárez M (2017) Website fingerprinting defenses at the application layer. In: Proceedings on privacy enhancing technologies, pp 186–203. https://doi.org/10.1515/popets-2017-0023
Cherubin G, Jansen R, Troncoso C (2022) Online website fingerprinting: Evaluating website fingerprinting attacks on tor in the real world. In: 31st USENIX security symposium, pp 753–770
Deng X, Yin Q, Liu Z et al (2023) Robust multi-tab website fingerprinting attacks in the wild. In: 2023 IEEE symposium on security and privacy (SP), pp 1005–1022, https://doi.org/10.1109/SP46215.2023.10179464
Dingledine R, Mathewson N, Syverson P (2004) Tor: the second-generation onion router. In: 13th USENIX security symposium, pp 303–319
Dyer KP, Coull SE, Ristenpart T et al (2012) Peek-a-Boo, i still see you: Why efficient traffic analysis countermeasures fail. In: 33rd IEEE symposium on security and privacy (SP), IEEE symposium on security and privacy, pp 332–346. https://doi.org/10.1109/sp.2012.28
Gong J, Zhang W, Zhang C et al (2022) Surakav: generating realistic traces for a strong website fingerprinting defense. In: 43rd IEEE symposium on security and privacy, pp 1558–1573. https://doi.org/10.1109/SP46214.2022.9833722
Gong JJ, Wang T (2020) Zero-delay lightweight defenses against website fingerprinting. In: 29th USENIX security symposium, pp 717–734
Goodfellow I, Pouget-Abadie J, Mirza M et al (2020) Generative adversarial networks. Commune ACM 63(11):139–144. https://doi.org/10.1145/3422622
Hayes J, Danezis G (2016) k-fingerprinting: a robust scalable website fingerprinting technique. In: 25th USENIX security symposium, pp 1187–1203
He KM, Zhang XY, Ren SQ et al (2016) Deep residual learning for image recognition. In: 2016 IEEE conference on computer vision and pattern recognition (CVPR), IEEE conference on computer vision and pattern recognition, pp 770–778. https://doi.org/10.1109/cvpr.2016.90
Henri S, Garcia-Aviles G, Serrano P et al (2020) Protecting against website fingerprinting with multihoming. In: Proceedings on privacy enhancing technologies, pp 89–110. https://doi.org/10.2478/popets-2020-0019
Holland JK, Hopper N (2022) Regulator: a straightforward website fingerprinting defense. In: Proceedings on privacy enhancing technologies, pp 344–362. https://petsymposium.org/popets/2022/popets-2022-0049.php
Juarez M, Imani M, Perry M et al (2016) Toward an efficient website fingerprinting defense. In: 21st european symposium on research in computer security (ESORICS), pp 27–46. https://doi.org/10.1007/978-3-319-45744-4_2
Kadianakis G, Polyzos T, Perry M et al (2021) Tor circuit fingerprinting defenses using adaptive padding. arXiv:2103.03831v2
Kwon A, AlSabah M, Lazar D et al (2015) Circuit fingerprinting attacks: passive deanonymization of tor hidden services. In: 24th USENIX security symposium, pp 287–302
Lecun Y, Bottou L, Bengio Y et al (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324. https://doi.org/10.1109/5.726791
Li S, Guo HJ, Hopper N (2018) Measuring information leakage in website fingerprinting attacks and defenses. In: ACM SIGSAC conference on computer and communications security (CCS), pp 1977–1992. https://doi.org/10.1145/3243734.3243832
Lu D, Bhat S, Kwon A et al (2018) Dynaflow: An efficient website fingerprinting defense based on dynamically-adjusting flows. In: 17th ACM workshop on privacy in the electronic society (WPES), pp 109–113. https://doi.org/10.1145/3267323.3268960
Mathews N, Sirinam P, Wright M (2018) Understanding feature discovery in website fingerprinting attacks. In, (2018) IEEE Western New York Image and Signal Processing Workshop (WNYISPW), 2018 IEEE Western New York Image and Signal Processing Workshop. WNYISPW 2018. https://doi.org/10.1109/WNYIPW.2018.8576379
Mathews N, Holland JK, Oh SE et al (2023) SoK: a critical evaluation of efficient website fingerprinting defenses. In: 2023 IEEE symposium on security and privacy, pp 344–361
Nasr M, Bahramali A, Houmansadr A (2021) Defeating dnn-based traffic analysis systems in real-timewith blind adversarial perturbations. In: 30th USENIX security symposium, pp 2705–2722
Nithyanand R, Cai X, Johnson R (2014) Glove: a bespoke website fingerprinting defense. In: Proceedings of the 13th workshop on privacy in the electronic society. Association for Computing Machinery, New York, pp 131–134. https://doi.org/10.1145/2665943.2665950
Panchenko A, Niessen L, Zinnen A et al (2011) Website fingerprinting in onion routing based anonymization networks. In: Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society. Association for Computing Machinery, pp 103–114. https://doi.org/10.1145/2046556.2046570
Panchenko A, Lanze F, Zinnen A et al (2016) Website fingerprinting at internet scale. In: 23rd annual network and distributed system security symposium (NDSS). https://doi.org/10.14722/ndss.2016.23477
Pulls T (2020) Towards effective and efficient padding machines for tor. arXiv:2011.13471
Rahman MS, Sirinam P, Mathews N et al (2020) Tik-tok: the utility of packet timing in website fingerprinting attacks. In: Proceedings on privacy enhancing technologies, vol 2020. De Gruyter, pp 5–24, https://doi.org/10.2478/popets-2020-0043
Rahman MS, Imani M, Mathews N et al (2021) Mockingbird: defending against deep-learning-based website fingerprinting attacks with adversarial traces. IEEE Trans Inf Forensics Secur 16:1594–1609. https://doi.org/10.1109/tifs.2020.3039691
Rimmer V, Preuveneers D, Juarez M et al (2018) Automated website fingerprinting through deep learning. In: 25th annual network and distributed system security symposium (NDSS). https://doi.org/10.14722/ndss.2018.23105
Rosenblatt M (1956) Remarks on some nonparametric estimates of a density function. Ann Math Stat 27(3):832–837. https://doi.org/10.1214/aoms/1177728190
Se Eun O, Mathews N, Rahman MS et al (2021) Gandalf: gan for data-limited fingerprinting. In: Proceedings on privacy enhancing technologies, pp 305–322. https://doi.org/10.2478/popets-2021-0029
Shen M, Ji K, Gao Z et al (2023) Subverting website fingerprinting defenses with robust traffic representation. In: 32nd USENIX security symposium. USENIX Association, pp 607–624
Shmatikov V, Wang MH (2006) Timing analysis in low-latency mix networks: attacks and defenses. In: 11th European symposium on research in computer security. https://doi.org/10.1007/11863908_2
Sirinam P, Imani M, Juarez M et al (2018) Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In: ACM SIGSAC conference on computer and communications security (CCS), pp 1928–1943. https://doi.org/10.1145/3243734.3243768
Sirinam P, Mathews N, Rahman MS, et al (2019) Triplet fingerprinting: more practical and portable website fingerprinting with n-shot learning. In: ACM SIGSAC conference on computer and communications security (CCS), pp 1131–1148. https://doi.org/10.1145/3319535.3354217
Smith JP, Dolfi L, Mittal P et al (2022) QCSD: a QUIC client-side website-fingerprinting defence framework. In: 31st USENIX security symposium, pp 771–789
Padding spec (2019) Tor padding specification. https://github.com/torproject/torspec/blob/main/padding-spec.txt/
Wang T (2021) The one-page setting: a higher standard for evaluating website fingerprinting defenses. In: ACM SIGSAC conference on computer and communications security (ACM CCS), pp 2794–2806. https://doi.org/10.1145/3460120.3484790
Wang T, Goldberg I (2017) Walkie-talkie: an efficient defense against passive website fingerprinting attacks. In: 26th USENIX Security Symposium. pp 1375–1390
Wang T, Cai X, Nithyanand R et al (2014) Effective attacks and provable defenses for website fingerprinting. In: 23rd USENIX security symposium, pp 143–157
Witwer E, Holland JK, Hopper N (2022) Padding-only defenses add delay in tor. In: Proceedings of the 21st workshop on privacy in the electronic society, pp 29–33. https://doi.org/10.1145/3559613.3563207

Publisher's Note