TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 14/14 pp 1148–1159
DOI: 10.26599/TST.2023.9010062
Volume 28, Number 6, December 2023

An Online Website Fingerprinting Defense Based on the Non-Targeted Adversarial Patch

Xiaodan Gu*, Bingchen Song, Wei Lan, and Ming Yang

Abstract: Website Fingerprinting (WF) attacks can extract side channel information from encrypted traffic to form a fingerprint that identifies the victim's destination website, even if traffic is sophisticatedly anonymized by Tor. Many offline defenses have been proposed and claimed to have achieved good effectiveness. However, such work is more of a theoretical optimization study than a technology that can be applied to real-time traffic in the practical scenario. Because defenders generate optimized defense schemes only if the complete traffic traces are obtained. The practicality and effectiveness are doubtful. In this paper, we provide an in-depth analysis of the difficulties faced in porting existing offline defenses to the online scenarios. And then the online WF defense based on the non-targeted adversarial patch is proposed. To reduce the overhead, we use the Gradient-weighted Class Activation Mapping (Grad-CAM) algorithm to identify critical segments that have high contribution to the classification. In addition, we optimize the adversarial patch generation process by splitting patches and limiting the values, so that the pre-trained patches can be injected and discarded in real-time traffic. Extensive experiments are carried out to evaluate the effectiveness of our defense. When bandwidth overhead is set to 20%, the accuracies of the two state-of-the-art attacks, DF and Var-CNN, drop to 10.83% and 15.49%, respectively. Furthermore, we implement the real-time patch traffic injection based on WFPadTools framework in the online scenario, and achieve a defense accuracy of 95.50% with 12.57% time overhead.

Key words: website fingerprinting; online defense; adversarial patch; traffic analysis

1 Introduction

Currently, web has become one of the most important applications of the Internet. According to the statistics as of May 2023[1], the number of global websites and users have reached 1.12 billion and 5.18 billion, respectively. Web transmits a large amount of users' private information, such as social information, bank card numbers, shopping preferences, etc. Therefore, when users visit websites, they are increasingly inclined to use some privacy enhancing technologies to protect the communication contents and communication relationships. However, even when using sophisticated anonymous communication technologies like Tor[2], which uses multiple layers of encryption, packet fixed-length encapsulation, and the multi-hop forwarding mechanism for multiple obfuscations, there is still a risk of privacy disclosure. For example, an attacker can use the Website Fingerprinting (WF) attacks to identify the sites visited by a user. In the WF attack model as shown in Fig. 1, the attacker has access to the link between the client and Tor entry node in a passive manner, which means that he can only record the network packets rather than modify, delay, drop, or decrypt them. The attacker imitates the victim to browse the target sites and extracts

• Xiaodan Gu, Bingchen Song, Wei Lan, and Ming Yang are with the School of Computer Science and Engineering, Southeast University, Nanjing 211189, China. E-mail: xdgu@seu.edu.cn; songbc@seu.edu.cn; 220181610@seu.edu.cn; yangming2002@seu.edu.cn.
* To whom correspondence should be addressed.
Manuscript received: 2023-06-05; revised: 2023-06-15; accepted: 2023-06-15

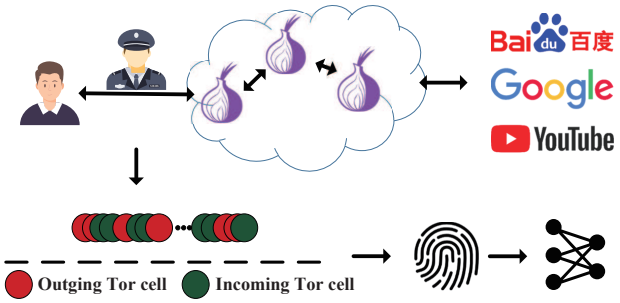

Fig. 1 WF threat model.

traffic features to form fingerprints through side-channel analysis techniques. Then a classifier is trained to predict a victim's activity. A large amount of work has been proposed to demonstrate the effectiveness[3–6].

In order to further enhance privacy, researchers propose various WF defenses[7–10], including injecting dummy packets[7] in the traffic sequence, loading background pages during the web access[8], sending packets at a constant rate and fixed size[9], and so on. They hope to use these methods to disrupt traffic characteristics. However, such defenses are either difficult to deploy in practice because of the excessive overhead or failed against the deep learning-based WF attacks[10]. With the continuous development of the adversarial sample techniques[11], they are introduced in WF defenses. However, such defenses require complete traffic traces to construct adversarial samples, and thus can only perform offline calculations. It makes these methods more of a theoretical optimization study and difficult to deploy effectively in practical scenarios.

In this paper, we propose an online WF defense based on the non-targeted adversarial patch. The main contributions of our work are as follows:

• We provide an in-depth analysis of the difficulties faced in porting existing offline defenses to the online scenarios.

• We propose an online WF defense based on the non-targeted adversarial patch. The Gradient-weighted Class Activation Mapping (Grad-CAM) algorithm is utilized to determine the critical segments. We also optimize the generation of the non-targeted adversarial patch.

• We implement the real-time injection of the pre-trained adversarial patch based on WFPadTools framework. By carrying out extensive experiments, we demonstrate the practicality and effectiveness of our attack.

We organize the rest of the paper as follows. We introduce related work in Section 2 and describe the problem in Section 3. We propose our online defense in Section 4. We evaluate our defense extensively in Section 5. Finally, we conclude our work in Section 6.

2 Related Work

2.1 WF attack

According to whether manual feature extraction is required, we can divide the existing WF attacks into traditional machine learning based methods and deep learning-based methods. The former use expert knowledge to extract effective features from network anonymous traffic to train various classifiers, such as objects' sizes, packet sizes, packet ordering, inter-arrival times, and their statistical values[8, 12–16]. However, with the continuous development of anonymity techniques and defense methods, many features are no longer applicable to current WF attacks. It is difficult to extract new effective features, and attack accuracy cannot be further improved. In recent years, the deep learning model has become a key technology in many fields due to its effectiveness. Researchers have also applied it to WF attacks, and many effective attacks are emerged. Sirinam et al.[10] proposed the Deep Fingerprinting (DF) attack, where they created a deeper and more complex Convolutional Neural Network (CNN) architecture by repeatedly stacking basic blocks consisting of the convolution, pooling, and activation layers. They experimented on a 95×1000 traffic sample dataset and achieved 98% accuracy in the closed world scenario. Rahman et al.[4] believed that the inter-arrival times can be used to improve the accuracy. They added temporal features of the burst traffic and use the same CNN architecture. Bhat et al.[6] suggested that packet sequences had more complex global relationships than images, and the beginning packets might have a larger ripple effect. They proposed the Var-CNN attack to capture features by using dilated causal convolution. The causal convolution can capture the contextual ordering features, while dilation convolution is able to obtain a large receptive field. They combined manual feature extraction as well as automatic feature extraction to improve the recognition accuracy. They achieved an accuracy of 97.8% on a small dataset of 100×100 traces.

2.2 WF defense

The WF defenses usually modify users' traffic to obfuscate traffic patterns and reduce the classification accuracy. They can be roughly divided as follows: traffic regularization based defenses, traffic camouflage based defenses, traffic obfuscation based defenses, and

adversarial defenses.

• **Traffic regularization based defenses**[9, 17]. These defenses shape the network traffic of all target websites into the same pattern to severely limit the feature space by adding constraints to the behaviors of sending and receiving packets. Dyer et al.[17] proposed the Buffered Fixed Length Obfuscation (BUFLO) defense method, which regularized traffic by sending packets at a constant rate and fixed size in both directions, and continuously sent dummy packets for a fixed period after the communication ends. Their defense reduced the accuracy of K Nearest Neighbors (KNN)[16] and K-Fingerprinting (K-FP)[3] methods to 10% and 21%, respectively. Although traffic regularization based defenses can largely reduce the performance of WF attacks, the resulting time and data overhead are too large to apply them in practical scenarios.

• **Traffic camouflage based defense**[18, 19]. Such methods disguise network traffic as target website by injecting dummy packets or delays to reduce the recognition accuracy of the classifier. Wang and Goldberg[18] proposed an efficient WF defense called Walkie-Talkie. It modified the communication mode to half-duplex to generate an easy-to-operate traffic sequence, and then decomposed the traffic sequence into a series of bursts. For each target page, they selected another decoy page and constructed a hyper-sequence for two pages to fool the classifier.

• **Traffic obfuscation based defenses**[7, 20]. These defenses randomly inject dummy packets or delays to make users generate different traffic patterns each time they visit the target website. Juarez et al.[7] proposed an adaptive defense called WTF-PAD, which aimed to disrupt the burst characteristics. It could detect if there was a large delay between consecutive bursts and used an adaptive algorithm to inject dummy packets to fill the gaps. This defense increased the data overhead by 54%, while the time overhead was 0. In addition, some server-side defenses were proposed. Lin et al.[20] studied the new mechanism Server Push of HTTP 2.0. They found that if the server was set to actively push all objects once an HTTP request is received, the accuracy of the DF[10] attack dropped to 74%.

• **Adversarial defenses**[21, 22]. For deep learning based WF attacks, researchers use the idea of adversarial attack to cause misclassification of the classifiers, by injecting tiny and adversarial perturbations into traffic. Hou et al.[21] proposed a defense WF-GAN based on Generative Adversarial Networks (GAN) network to generate adversarial samples. They divided the sites into the source site set and the target site set, and trained the generator to inject perturbations to the source sites' traffic, making its characteristics similar to the target sites. The experimental results showed that the WF-GAN defense made the accuracy of DF down to 10%. Gong et al.[22] proposed Surakav, which made use of a GAN to generate different sending patterns and regulated buffered data according to the sampled patterns. Although the adversarial defenses can greatly reduce the performance of WF attacks with lower bandwidth overhead, they need to generate adversarial samples based on the complete traffic traces, which makes less practicality and effectiveness in the online scenario.

3 Problem

3.1 Problem analysis

Given the great success of deep learning models in WF attacks, the state-of-the-art defenses are almost focused on how to fool models with traffic obfuscation. They always inject random dummy packets and delays into the original flows to change the traffic patterns. However, such approaches need to train perturbation models based on the target websites' complete traffic traces offline, resulting in low possibility of being put into practice. When we explore the practicalities of online WF defenses, two problems are faced as follows:

• **There is no guarantee that the offline optimal perturbations will be effective in the online scenario.**

In the offline scenario, WF defenders calculate the optimal perturbations based on the complete historical traffic traces and inject the corresponding dummy traffic into the dataset. However, due to network jitter, site updates, and other factors, the generated network traffic changes in sequence patterns even when the same web page is accessed repeatedly. As shown in Fig. 2, the packet sequences generated by two visits on Site W has changed. In addition, injected dummy traffic may trigger packet loss or data retransmission, deviating the obfuscated traffic pattern from the theoretical model. So, it is likely to lead to poor effectiveness of applying historical perturbations to the real-time traffic.

• **Defenders can not foresee the sequence pattern of upcoming traffic.**

Assuming that an optimal perturbation strategy is currently in place, i.e., we know what to disguise the

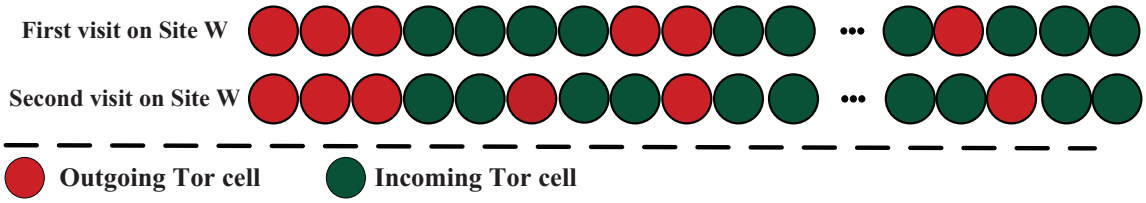

Fig. 2 Packet sequences generated by two visits on Site W.

traffic pattern as, the situation is still not promising. This is because in the online scenario, as shown in Fig. 3, the defender can only observe the sequence of packets generated before the current time point and cannot foresee the upcoming network traffic pattern, which leads to an inability to efficiently perform the injection of the dummy traffic, making the defense less effective.

3.2 Defense model

To improve the feasibility of the online WF defense, we propose a defense model as shown in Fig. 4. The defender can obtain historical traffic of target sites for training. He also has white-box access to the classification model used by the attacker, including the architecture and parameters. For the sake of traffic obfuscation in real-time, the defender needs to control both the client and bridge node, so that he can inject or remove dummy packets and delays.

4 Methodology

In this section, we first present the basic idea of our online WF defense, then introduce each step of the workflow in detail.

4.1 Basic idea

By analyzing the problems faced by online WF defenses, we can conclude that there are two requirements for perturbations of original traffic.

• The traffic perturbations should be robust against changes in original traffic sequences of target websites.

• The traffic perturbations should preferably be position-independent, meaning that the defender can inject dummy packets and delays at any time and any sequence position without significantly affecting the defense effect.

We choose to use the adversarial patch technology[23] to fix the aforementioned two problems. Adversarial patch technology was initially proposed by researchers in the field of image processing and belongs to one type of adversarial attacks. However, unlike traditional adversarial perturbation methods, defenders are not limited to injecting tiny and imperceptible perturbations that are specific to the target image sample. Instead, they focus on generating a universal adversarial patch. Once the patch is generated, it can be placed on any location of any image sample, resulting in the classifier predicting the desired class. The fundamental reason is that the added patch has higher discriminability compared to the original image sample. In other words, the pattern of the adversarial patch is easier for the classifier to learn and identify, resulting in scrambled images being wrongly classified into the category to which the adversarial patch belongs.

To obtain the trained patch P_{adv}, the defender first determines the target class and then optimizes the objective function as shown in Eq. (1).

$$P_{\text{adv}} = \arg\min_{p} E_{v\in V, l\in L}[\text{loss}\,(F\,(G\,(p, v, l)), y_t)] \tag{1}$$

where V is the image dataset, L is the injection location, and G is the injection function that can be used to inject the patch p into image v at location l. F is the classifier used to predict the classes of images, and the expectation E is used to improve the trained patch's robustness regardless of what is in the background. By minimizing

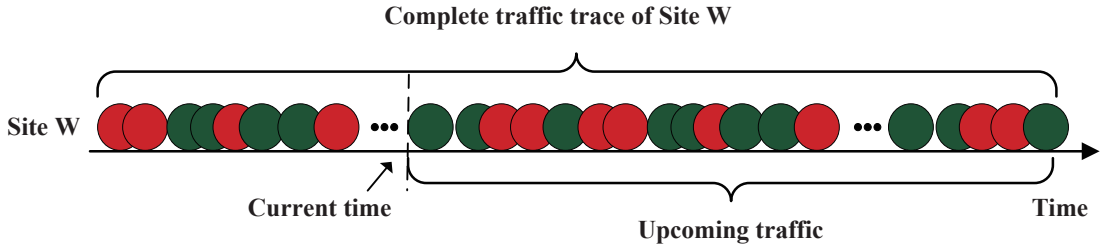

Fig. 3 Traffic trace observed by the defender.

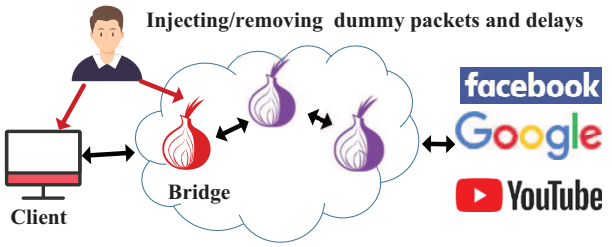

Fig. 4 Defense model.

the loss between the output of F and the target class y_t for all images in V, the trained patch P_{adv} can be effective to fool the classification model.

From the above analysis, it is clear that the trained adversarial patch is independent of input images, meaning that no matter what image is inputted, it will result in incorrect classification. Therefore, the adversarial patch technique may have better robustness against different traffic sequences generated from the same site, allowing defenders to pre-calculate patches without knowing the input traffic traces. Furthermore, the patch can be injected at any position, which satisfies another requirement for online WF defenses.

4.2 Overview of the online WF defense

We propose an online WF defense based on non-targeted adversarial patches and the overall workflow can be divided into three steps.

• **Step 1: Identification of critical segments.** We divide the traffic sequence generated by the target website into multiple segments and calculate the contribution of each segment to the classification result. And the segments with higher contribution are called critical segments. Identifying the key areas can help us to find appropriate injection positions and reduce the data overhead.

• **Step 2: Generation of the adversarial patch.** By analyzing the differences between online and offline defense scenarios, we improve the traffic adversarial patch generation method by introducing several constraints.

• **Step 3: Online injection of the adversarial patch.** We make a Tor plug-in based on the WFPadTools[24] framework to inject the trained patch into the real-time traffic of the target site.

4.3 Identification of critical segments

In the field of image processing, the adversarial patch can be position-independent because it ensures that the generated patch is sufficiently distinguishable compared to the original image samples. When we apply it to the network traffic, a high bandwidth and latency overhead is required to improve the discriminability of the traffic patch. To reduce the overhead, we attempt to decrease the discriminability of the original traffic trace rather than increasing that of the traffic patch, i.e., to destroy the sequence pattern of the critical traffic segments.

We build a classifier using the convolutional neural network, which is the most typical deep learning recognition model in existing work. The classification model consists of three convolutional groups, each of which includes a convolutional layer, a batch normalization layer, and an activation layer. Finally, a fully connected layer is used to output the result. This model identifies the websites by sliding the convolutional kernel to match specific sequence patterns of traffic. So we utilize the Grad-CAM technique[25] to measure the contribution degree to the model's prediction of each segment in the traffic sequence, as shown in Fig. 5.

We first input the network packet sequence into the convolutional neural network to obtain the output (denoted by A) of the last convolutional layer and the score of the target class label c without softmax (denoted by y_c). A contains N feature maps, each of size $Z = 1 \times I$, and A_k^i denotes the value of the i-th row in the k-th feature map. Then the gradient of each value in the k-th feature map (denoted by g_k^c) is calculated using back propagation, and we can obtain the weight value of the k-th feature map by averaging all gradient values, as shown in Eqs. (2) and (3). This process is equivalent to performing a global average pooling operation on all feature maps.

$$g_k^c = \frac{\partial y_c}{\partial A_k^i} \tag{2}$$

$$a_k^c = \frac{1}{Z}\sum_i g_k^c \tag{3}$$

Once the weights of all feature maps are obtained, the contribution degree of each point (denoted by f^c) can be calculated by multiplying A_k with the corresponding weight a_k^c and then adding them together.

$$f^c = \sum_k a_k^c A_k \tag{4}$$

Then the ReLU operation is performed on the f^c to obtain L^c, which keeps out only the points that have a positive effect on the category c. Moreover, L^c is divided into several groups, and the group contribution degree is obtained by summing up values of all points in the group. Finally, by comparing the group contribution degrees, we can identify the critical traffic segments.

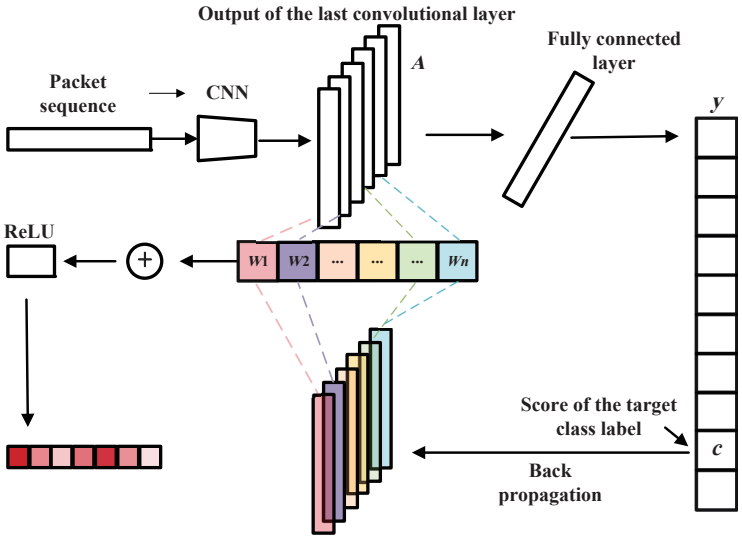

Fig. 5 Identification of critical segments.

4.4 Generation of the adversarial patch

The adversarial image patches are injected by overlaying the patch on the original image in the form of pixel value replacement. However, all network packets are encrypted. If we replace the packets, website access errors will occur. As a result, we can inject the traffic patch only by delaying the original packets and inserting dummy cells, as shown in Fig. 6.

The traditional adversarial patch is usually a complete image that has been added to any original samples. However, in the online scenario, if we continuously delay original packets and insert many dummy packets at a certain position, it will cause severe packet loss and retransmissions. Therefore, we split the traffic patch into smaller patch blocks and inject them at different positions so that they can perturb patterns of the corresponding segments, as shown in Fig. 7.

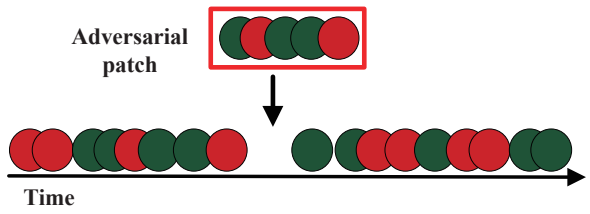

Fig. 6 Injection method of the adversarial patch.

Furthermore, we also need to make some restrictions on the patch generation process. Since the anonymous packets are represented as $+1$ or -1 depending on the upstream and downstream directions, the patch element is also assigned $+1$ or -1. In addition, the operation of patch injection needs to be done on both the Tor client and bridge node in the online scenario. In our design, we obfuscate traffic from the client's point of view, and the client needs to send control messages to instruct the bridge node to perform operations. Before patch injection, the client creates a handshake with the bridge and then sends patch information. The corresponding first packet from the bridge is considered to be the start of the patch traffic. So, the first element of each small patch block is set to -1. Similarly, at the end of the patch injection, the client needs to send a control message to inform the end of injection. Therefore, the last element of each small patch block needs to be set to $+1$. The adversarial traffic patch is trained to optimize the objective function

$$P_{\text{adv}} = \arg\max_{p} E_{x \in X}[\text{loss}(F(\psi(p, x, l)), y_t)] \quad (5)$$

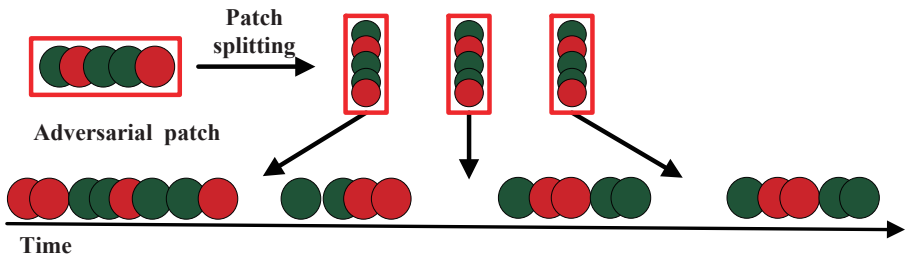

Fig. 7 Process of patch splitting.

where X is the original traffic dataset, l is the critical segment, and ψ is the injection function. F is the classifier while y_t indicates the class label. We use cross-entropy as the loss function and maximize F's classification loss to update the traffic patch.

We summarize the generation of the adversarial traffic patch in Algorithm 1. In each iteration, for each original traffic trace, an adversarial patch P is injected at the position S. Specifically, for each selected critical segment, we randomly select a position within the segment to inject a small patch block. Since there is no guarantee that the small patch blocks will be injected precisely to the specified positions in the online scenario, the random selection method enhances robustness. Then we calculate the loss value and back propagates it using Stochastic Gradient Descent (SGD). To apply SGD, we relax the restriction during the patch optimization by allowing each element to be a continuous value between $[-1, 1]$. When the optimization is finished, all elements greater than 0 are set to $+1$ and the opposite are set to -1. Finally, by continuously iterating, the adversarial patches for each site are generated.

4.5 Injection of the adversarial patch

Once the adversarial patch is generated, we need to inject it into the real-time packet sequence at the selected positions. So, we make a Tor plug-in based on the WFPadTools framework to implement the patch loading and injection. WFPadTools provides the necessary building modules for the development of link-padding based WF strategies in Tor. It allows one side to inject dummy messages and discard them at the other side. Furthermore, WFPadTools defines a new protocol between the Tor protocol layer and TCP layer. The message formats are shown in Figs. 8 and 9, where data messages are used to transmit real data or dummy data, and control messages are used to send control information.

The length of the data message's header is 5 bytes, which consists of the Total Length field (2 bytes), Payload Length field (2 bytes), and Flag field (1 byte). The values of the Flag field are shown in Table 1. When the Total Length is set to 514, the Payload Length is set to 0, and the Flag is set to 1, it will generate a dummy cell.

The control message contains the same header fields as the data message, in addition to the Opcode and Args fields, which allow the operation code and parameters to be specified. We customize Opcode and Args to implement the control commands between the client and bridge node, and the specific parameter values are shown in Table 2.

The adversarial patch injection process is shown in Fig. 10. For both the client and bridge, they have two states, the normal state and injection state. In the normal state, both sides follow Tor's communication process for sending and receiving real data messages. Vice versa, they delay and store real data in the buffer and send padding cells. For instance, when a user initiates an HTTP request for the target site, the Tor client in the normal state first loads the corresponding pre-trained adversarial patch. Then it records the direction of each

Algorithm 1 Adversarial patch generation algorithm

Input: Group contribution degree D, small patch block length pb_{len}, bandwidth overhead (in percentages) B, minimum sequence length $\text{minseq}_{\text{len}}$, original traffic trace dataset X_w, iteration number T, and classifier F, and loss function Loss.

Output: Adversarial traffic patch P and injection position S.

Calculate the number of small patch blocks $N = \frac{\text{minseq}_{\text{len}} \times B}{\text{pb}_{\text{len}}}$
Initialize the adversarial patch P and split it into N small patch blocks
Select top N critical segments and get the injection position $S = \{s_1, s_2, \ldots, s_N\}$
for EPOCH = 1 to T **do**
  for x in X_w **do**
    $X \leftarrow \text{InsertPatch}(x, P, S)$
    $\text{pre} \leftarrow F(x)$
    $\text{Loss} \leftarrow \text{Loss}(\text{pre}, y_x)$
    Update P by Loss.backword()
  end for
end for

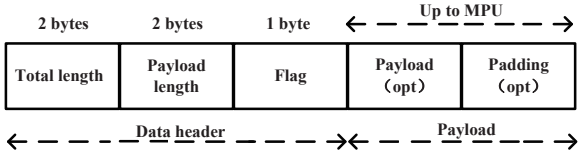

Fig. 8 Data message format.

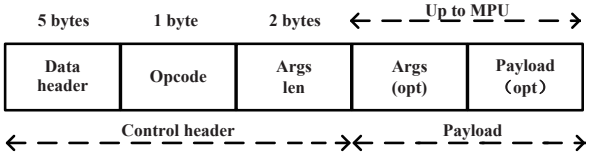

Fig. 9 Control message format.

Table 1 Values of the Flag field.

Value	Description
0	Real data
1	Padding data need to be dropped
2	Control information is stored in the payload

Table 2 Values of the Opcode and Args fields.

Opcode	Type and description	Arg	Direction
0	PADDING_START (Indicate the beginning of the injection)	The number of padding cells sent from the bridge	Client→ Brigde
1	PADDING_STARTED (Response to PADDING_START)	–	Bridge→Client
2	SEND_PADDING (Send the padding data)	The number of padding cells sent from the bridge	Client→ Brigde
3	PADDING_END (Indicate the end of the injection)	–	Client→ Brigde

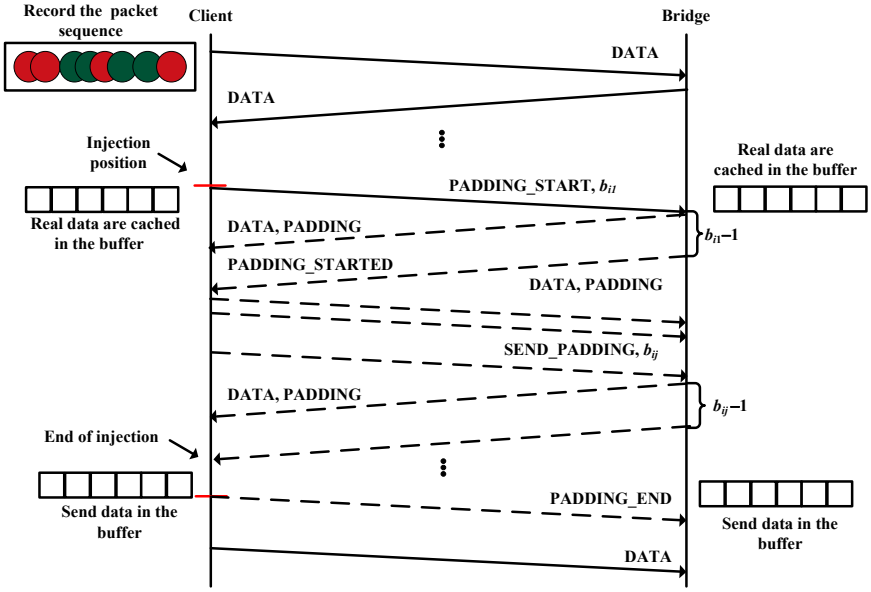

Fig. 10 Adversarial patch injection process.

packet sent or received to generate a real-time packet sequence. When the i-th injection position is reached, the client switches to the injection state. It caches subsequent outgoing packets into the buffer, and send the PADDING_START message to the bridge, which also indicates the padding number of the first burst b_{i1} should be send from the bridge. Once the bridge receives the PADDING_START message, it changes to the injection state and sends $b_{i1} - 1$ dummy cells plus the PADDING_STARTED message, which means the first busrt traffic of the i-th small patch block has been sent. Then the client continues to send dummy packets according to the trained adversarial patch. When all the traffic of the i-th small patch block is injected, the client sends PADDING_END message, switches to the normal state, and then sends real data cached in the buffer. The same operations are performed when the bridge receives the PADDING_END message. Now, the injection of the i-th small patch block is completed. For each small patch block, repeat the above operations. Finally the adversarial patch can be successfully injected.

5 Evaluation

In this Section, we implement the proposed online WF defense to evaluate the effectiveness and efficiency.

5.1 Experimental setup

To avoid interference from other traffic in the system, we package the data collection program as a Docker image and deployed it on 10 servers, and deploy 5 additional Tor bridge nodes. The server system is Ubuntu 18.04 with Linux kernel version 4.15.0-175, Docker version 20.10.14, and Tor version 0.4.5.10. We use the automation tool Selenium to launch TorBrowser with the version 10.5.10 to simulate user's website access behavior. In addition, the collection program will determine whether the webpage is successfully accessed based on the HTTP status code. If the page is successfully accessed, it will save the traffic sequence and the screenshot.

Similar to previous work, we turn off cache. We also disable some browser features which may generate noise traffic, such as automatic updates and speculative pre-connections. When deploying the defense policy online, we modify the Onion Proxy (OP) and bridge's configuration in the Tor configuration file as shown in Table 3 to use a custom plug-in in both sides.

Table 3 OP and bridge's configuration file.

Tor entity	Torrc configuration file
OP	SocksPort 4997 Exitpolicy reject *:* UseBridges 1 ClientTransportPlugin MLpatch exec /wfpadtools/bin/obfsproxy \ –log-min-severity=debug \ –log-file=/root/mlpatch_client/server.log \ managed Bridge MLpatch 45.77.71.64:40535
Bridge	AssumeReachable 1 PublishServerDescriptor 0 Exitpolicy reject *:* ORPort 4053 BridgeRelay 1 ServerTransportListenAddr MLPatch 0.0.0.0:40535 ServerTransportPlugin MLPatch exec /wfpadtools/ bin/obfsproxy\ –log-min-severity=debug \ –log-file=/root/mlpatch_server/server.log \ managed

5.2 Data collection

We Select 100 most popular websites from the Alexa, which ranks all sites in the Internet according to the number of visitors, links, etc. For the domains of different countries or regions of the same website, we unify them, i.e., Google.com and Google.com.hk are unified as Google.com. We generate two datasets collected called Undefended_traffic and Defended_traffic, which are shown in Table 4.

For Undefended_traffic dataset, the collection program visits 100 target websites respectively, 600 times each and finally 60 000 traffic instances are collected. When the collection is completed, the blank pages' instances were removed according to the screenshots, and then the sites with insufficient numbers were supplemented. Websites with insufficient instances will be revisited. For Defended_traffic dataset, we first deploy the online defense policy and collect traffic data by visiting 100 target websites, each 100 times. It is important to note that Defended_traffic dataset is used as a test set for the online WF defense.

5.3 Evaluation indicator

We propose 3 evaluation indicators as follows:

Table 4 Traffic trace dataset.

Name	Size
Undefended_traffic	100×600
Defended_traffic	100×100

• **Defense Accuracy (DACC)**. We use the defense accuracy DACC to evaluate the defense capability of the proposed method, which represents the probability that the classifier misidentifies.

$$\text{DACC} = 1 - \text{Accuracy} \tag{6}$$

• **Bandwidth Overhead (BWOH)**. Bandwidth overhead represents the additional data overhead caused by the defense. Note that padding packets are dropped on the bridge and do not affect the web server.

$$\text{BWOH} = \frac{N_P}{N_O} \times 100\% \tag{7}$$

where N_P is the number of dummy packages, and N_O is the number of original packets generated by the target website.

• **Time Overhead (TOH)**. The time overhead is calculated as shown in Eq. (8). A large time overhead may degrade the customer experience.

$$\text{TOH} = \frac{T_E}{T_O} \times 100\% \tag{8}$$

where T_E represents the additional time generated by the defense, while T_O is the time required for the normal access to the target site.

5.4 Experimental result

For two datasets, we represent each instance as a time-directional sequence with a fixed length of 5000 packets. If some instance's length is less than 5000, we fill 0 at the end of the sequence. Conversely, truncate for sequences longer than 5000.

5.4.1 Identification of critical segments

We use Undefended_traffic dataset to identify the critical segments of each site. Firstly, we use the Grad-CAM method to generate the contribution degree of each point in the traffic sequence. Then we divide the sequence into 100 groups with the length of 50 packets, and obtain the group contribution degrees. Take the site chaturbate.com as an example, the visual heat map of its group contribution degrees is shown in Fig. 11.

We can see that the contribution degrees of each group are not exactly the same, and most critical segments are located at the head and tail of the sequence, while the middle region contributes less to the classification.

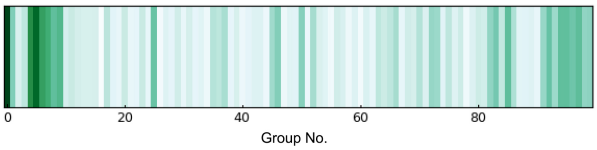

Fig. 11 Heat map of the group contribution degrees for chaturbate.com.

Therefore, when we perform the injection of the adversarial patch, identifying critical segments helps mislead the classifier.

5.4.2 Offline experiment results

We also use Undefended_traffic dataset to perform offline experiments. The dataset is divided into training set and test set in the ratio of 5 to 1. Then we use the training set to generate a traffic patch for each site. Finally, the trained patches are applied to the test set to evaluate the defense effectiveness.

In our method, there are two adjustable parameters, which are the length of the small patch block pb_{len} and the bandwidth overhead BWOH. The former determines the number of dummy packets for each critical segment, and the latter determines the total number of dummy packets. They all affect the defense effectiveness of the adversarial patch. For the same bandwidth overhead, the length of the small patch block is inversely proportional to the number. Obviously, the longer the small patch block length, the greater the impact on each critical segment. And the greater the number of small patch blocks, the greater the area of influence. It is worth noting that the length of the small patch block should not be too large, otherwise it will cause excessive delay and generate problems such as packet loss and TCP retransmission.

We set BWOH to 10%, 20%, and 30%, respectively, and perform experiments with different small patch block lengths. The results are shown in Fig. 12. We can see that when BWOH is 30% and pb_{len} is 30, the highest DACC of 95.79% is achieved. When pb_{len} exceeds 30, DACC decreases as pb_{len} increases. Therefore, it is necessary to compromise BWOH and pb_{len} to achieve a higher DACC. In this paper pb_{len} is set to 30.

Then we perform experiments to find the optimal value of BWOH, and the results are shown in Fig. 13. We can

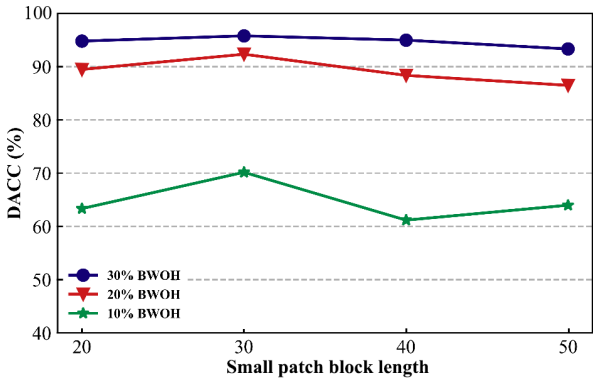

Fig. 12 DACC of different small patch lengths with different bandwidth overheads.

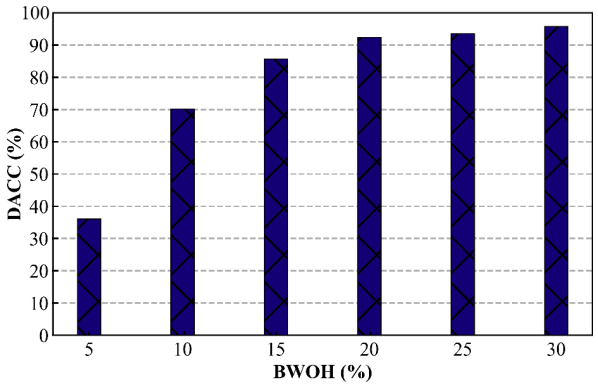

Fig. 13 DACC with different values of BWOH.

see that DACC increases as BWOH increases. When BWOH reaches 20%, DACC can already reach 92.34%. When BWOH continues to grow, DACC does not grow significantly.

Therefore, we set BWOH to 20% in order not to bring too much bandwidth consumption. Finally, the offline experimental results are as shown in Table 5. The results show that under the same conditions, random positions injection obtains 87.45% DACC, while it reaches 92.34% by injecting into critical segments. So, we can infer that identifying critical segments can improve the defense effectiveness.

We also evaluate the defense effectiveness against two classic WF attacks, DF and Var-CNN. The results are shown in Fig. 14. When using the proposed defense method, the accuracies of DF and Var-CNN models decrease significantly, which are 10.83% and 15.49%, respectively. It can be shown that our defense method is independent of the classifier used by the attacker.

Table 5 Offline experimental results.

Injection position	DACC (%)
Critical segments	92.34
Random positions	87.45

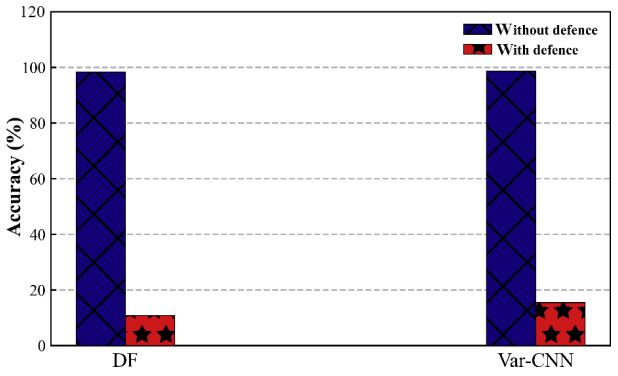

Fig. 14 Defense effect on different classification models.

5.4.3 Online experiment results

In the online scenario, we also set BWOH to 20% and pb_{len} to 30. The adversarial patches pre-trained in the offline scenario for target websites are saved in a file. When the client visits some website, the corresponding adversarial patch will be loaded. All the dummy packets are injected into the real-time traffic. The generated dataset is called Defended_traffic which contains 100×100 records. Then the attacker's classifier is applied to this dataset to evaluate the effectiveness of our defense in the online scenario. We obtain 95.50% DACC, which is better than the offline scenario. We try to explain the reason. When dummy cells are injected into real-time traffic, we need to delay some real packets, which will change the packet order and further reduce the accuracy of the classifier. Meanwhile the TOH is only 12.57%, which does not have a significant impact on the user experience.

6 Conclusion

In the previous work, numerous of WF defenses have been proposed to enhance the users' privacy, especially by introducing adversarial learning techniques to defend against deep learning based WF attacks. However, such defenses require complete traffic traces to construct adversarial samples, and thus can only perform offline calculations. In this paper, we first provide an in-depth analysis of the difficulties faced in porting existing offline defenses to the online scenarios. Then an online WF defense based on the non-targeted adversarial patch is proposed. Aiming at improving the practicality of defense in online scenarios, the Grad-CAM algorithm is utilized to determine the critical segments. We also optimize the generation of the non-targeted adversarial patch. Furthermore, we implement the real-time patch traffic injection based on WFPadTools framework. Extensive experiments are carried out to evaluate the effectiveness of our defense. In the online scenario, when bandwidth overhead is set to 20%, we have achieved a defense accuracy of 95.50% with 12.57% time overhead. In our future work, we will work on improving the robustness of the defense method without knowing the attacker's classification model.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China (Nos. 62102084 and 62072103), Jiangsu Provincial Natural Science Foundation of China (No. BK20190340), Jiangsu Provincial Key R&D Program (Nos. BE2021729, BE2022680, and BE2022065-4), Jiangsu Provincial Key Laboratory of Network and Information Security (No. BM2003201), and Key Laboratory of Computer Network and Information Integration of Ministry of Education of China (No. 93K-9).

References

[1] N. Huss, How many websites are there, https://siteefy.com/how-many-websites-are-there/, 2023.

[2] R. Dingledine, N. Mathewson, and P. Syverson, Tor: The second-generation onion router, in *Proc. 13th Conf. USENIX Security Symp.*, Berkeley, CA, USA, 2004, pp. 303–320.

[3] J. Hayes and G. Danezis, K-fingerprinting: A robust scalable website fingerprinting technique, in *Proc. 25th USENIX Conf. Security Symp.*, Austin, TX, USA, 2016, pp. 1187–1203.

[4] M. S. Rahman, P. Sirinam, N. Matthews, K. G. Gangadhara, and M. Wright, Tik-tok: The utility of packet timing in website fingerprinting attacks, arXiv preprint arXiv: 1902.06421, 2019.

[5] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen, Automated website fingerprinting through deep learning, arXiv preprint arXiv: 1708.06376, 2017.

[6] S. Bhat, D. Lu, A. Kwon, and S. Devadas, Var-CNN: A data-efficient website fingerprinting attack based on deep learning, arXiv preprint arXiv: 1802.10215, 2018.

[7] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright, Toward an efficient website fingerprinting defense, in *Proc. 21st European Symposium on Research in Computer Security*, Heraklion, Greece, 2016, pp. 27–46.

[8] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, Website fingerprinting in onion routing based anonymization networks, in *Proc. 10th Annual ACM Workshop on Privacy in the Electronic Society*, Chicago, IL, USA, 2011, pp. 103–114.

[9] X. Cai, R. Nithyanand, and R. Johnson, CS-BuFLO: A congestion sensitive website fingerprinting defense, in *Proc. 13th Workshop on Privacy in the Electronic Society*, Scottsdale, AZ, USA, 2014, pp. 121–130.

[10] P. Sirinam, M. Imani, M. Juarez, and M. Wright, Deep fingerprinting: Undermining website fingerprinting defenses with deep learning, arXiv preprint arXiv: 1801.02265, 2018.

[11] I. J. Goodfellow, J. Shlens, and C. Szegedy, Explaining and harnessing adversarial examples, arXiv preprint arXiv: 1412.6572, 2014.

[12] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine, Privacy vulnerabilities in encrypted HTTP streams, in *Proc. 5th Int. Conf. Privacy Enhancing Technologies*, Cavtat, Croatia, 2005, pp. 1–11.

[13] M. Liberatore and B. N. Levine, Inferring the source of encrypted HTTP connections, in *Proc. 13th ACM Conf. Computer and Communications Security*, Alexandria, VA, USA, 2006, pp. 255–263.

[14] D. Herrmann, R. Wendolsky, and H. Federrath, Website fingerprinting: Attacking popular privacy enhancing

technologies with the multinomial naïve-bayes classifier, in *Proc. 2009 ACM Workshop on Cloud Computing Security*, Chicago, IL, USA, 2009, pp. 31–42.

[15] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, Touching from a distance: Website fingerprinting attacks and defenses, in *Proc. 2012 ACM Conf. Computer and Communications Security*, Raleigh, NC, USA, 2012, pp. 605–616.

[16] T. Wang and I. Goldberg, Improved website fingerprinting on Tor, in *Proc. 12th ACM Workshop on Workshop on Privacy in the Electronic Society*, Berlin, Germany, 2013, pp. 201–212.

[17] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, Peek-a-boo, I still see you: Why efficient traffic analysis countermeasures fail, in *Proc. 2012 IEEE Symp. on Security and Privacy*, San Francisco, CA, USA, 2012, pp. 332–346.

[18] T. Wang and I. Goldberg, Walkie-Talkie: An efficient defense against passive website fingerprinting attacks, in *Proc. 26th USENIX Security Symp.*, Vancouver, Canada, 2017, pp. 1375–1390.

[19] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, Effective attacks and provable defenses for website fingerprinting, in *Proc. 23rd USENIX Conf. Security Symp.*, San Diego, CA, USA, 2014, pp. 143–157.

[20] W. Lin, S. Reddy, and N. Borisov, Measuring the impact of HTTP/2 and server push on web fingerprinting, in *Proc. Workshop on Measurements Attacks and Defenses for the Web* (*MADWeb*), San Diego, CA, USA, 2019, pp. 1–7.

[21] C. Hou, G. Gou, J. Shi, P. Fu, and G. Xiong, WF-GAN: Fighting back against website fingerprinting attack using adversarial learning, in *Proc. 2020 IEEE Symp. on Computers and Communications* (*ISCC*), Rennes, France, 2020, pp. 1–7.

[22] J. Gong, W. Zhang, C. Zhang, and T. Wang, Surakav: Generating realistic traces for a strong website fingerprinting defense, in *Proc. 2022 IEEE Symp. on Security and Privacy* (*SP*), San Francisco, CA, USA, 2022, pp. 1558–1573.

[23] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, Adversarial patch, arXiv preprint arXiv:1712.09665, 2017.

[24] WFPadTools, Framework to develop padding strategies on Tor Pluggable Transports, https://github.com/mjuarezm/wfpadtools, 2018.

[25] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, Grad-CAM: Visual explanations from deep networks via gradient-based localization, in *Proc. 2017 IEEE Int. Conf. Computer Vision* (*ICCV*), Venice, Italy, 2017, pp. 618–626.

Xiaodan Gu received the PhD degree in computer science from Southeast University, China, in 2018. Currently, she is a lecturer at the School of Computer Science and Engineering, Southeast University, Nanjing, China. Her research interests include network security and anonymous communication.

Bingchen Song received the BS degree in computer science from Southeast University, China, in 2021. He is currently pursuing the MS degree at the School of Computer Science and Technology, Southeast University, China. His research interests include network security, traffic analysis, and IoT.

Wei Lan received the MS degree in computer science from Southeast University, China in 2022. His research interests include network security and privacy.

Ming Yang received the PhD degree in computer science from Southeast University, China, in 2007. Currently, he is a professor at the School of Computer Science and Engineering, Southeast University, China. His research interests include network security and privacy.