A Multi-tab Website Fingerprinting Attack

Yixiao Xu[1], Tao Wang[2], Qi Li[1], Qingyuan Gong[3], Yang Chen[3], Yong Jiang[1]
[1]Graduate School at Shenzhen & Department of Computer Science, Tsinghua University, China
[2]Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong
[3]School of Computer Science, Fudan University, China
{xu-yx16@mails,qi.li@sz,yjiang@sz}.tsinghua.edu.cn, taow@cse.ust.hk, {chenyang,gongqingyuan}@fudan.edu.cn

ABSTRACT

In a Website Fingerprinting (WF) attack, a local, passive eavesdropper utilizes network flow information to identify which web pages a user is browsing. Previous researchers have extensively demonstrated the feasibility and effectiveness of WF, but only under the strong Single Page Assumption: the network flow extracted by the adversary always belongs to a single page. In other words, the WF classifier will never be asked to classify a network flow corresponding to more than one page, or part of a page. The Single Page Assumption is unrealistic because people often browse with multiple tabs. When this happens, the network flow induced by multiple tabs will overlap, and current WF attacks fail to classify correctly.

Our work demonstrates the feasibility of WF with the relaxed Single Page Assumption: we can attack a client who visits more than one pages simultaneously. We propose a multi-tab website fingerprinting attack that can accurately classify multi-tab web pages if they are requested and sequentially loaded over a short period of time. In particular, we develop a new BalanceCascade-XGBoost scheme for an attacker to identify the start point of the second page such that the attacker can accurately classify and identify these multi-tab pages. By developing a new classifier, we only use a small chunk of packets, i.e., packets between the first page's start time to the second page's start time, to fingerprint website. Our experiments demonstrate that in the multi-tab scenario, WF attacks are still practically effective. We have an average TPR of 92.58% on SSH, and we can also averagely identify the page with a TPR of 64.94% on Tor. Specially, compared with previous WF classifiers, our attack achieves a significantly higher true positive rate using a restricted chunk of packets.

CCS CONCEPTS

• **Security and privacy → Domain-specific security and privacy architectures;** • **Networks → Network privacy and anonymity;**

KEYWORDS

Website fingerprinting attack, Machine learning

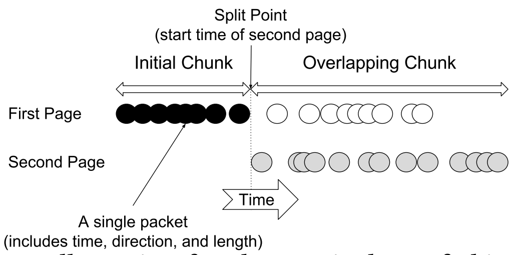

Figure 1: Illustration for the terminology of this paper, where black circles are the non-overlapped packets of the first page, white circles are overlapped packets of the first page, and grey circles are the packets of the second page.

ACM Reference Format:
Yixiao Xu, Tao Wang, Qi Li, Qingyuan Gong, Yang Chen, Yong Jiang. 2018. A Multi-tab Website Fingerprinting Attack. In *2018 Annual Computer Security Applications Conference (ACSAC'18), December 3-7, 2018, San Juan, Puerto Rico, USA.* ACM, NewYork, NY, USA, 15 pages. https://doi.org/10.1145/3274694.3274697

1 INTRODUCTION

When a client is browsing the web, she inevitably reveals her destination website to all on-path routers. ISPs who are running these routers may passively observe and collect clients' information for profit or due to legal pressure. Privacy enhancing technologies, such as Tor, can protect the client from such threats by encrypting her network flow and hiding the true source and destination through proxies. Even then, an attacker can still compromise a client's privacy by observing patterns in the network flow without decrypting them, using a technique known as Website Fingerprinting (WF). A site may prove to be uniquely identifiable from the order, number, size, and direction of the transferred network flow.

Recently, several studies have demonstrated the effectiveness of WF attacks [6, 11, 21]. However, Juarez et al. [14] criticized these works for overestimating the attacker's abilities. They highlighted the following critical assumption in all previous WF works, which we refer to as the Single Page Assumption: "The attacker knows when each web page starts loading and when it ends." Unfortunately, in practice, the assumption does not always hold [14, 23, 27]. In particular, users may want to open multiple tabs in their browsers, e.g., for the purpose of prefetching pages. Juarez et al. showed that without the assumption, WF attacks are highly inaccurate; yet, the Single Page Assumption remains unresolved.

In this paper, we propose a new WF attack that relaxes the Single Page Assumption. What allows the attack to succeed in addressing the Single Page Assumption is that we can split sequential multi-tab pages and classify accurately only the small chunk of packets of the first pages, e.g., even only using two seconds of data. Figure 1 illustrates the terminology and scenario of this paper. A client loads multiple pages simultaneously, with a small time gap between their start times. The first page is loaded, and after some time (corresponding to the split point), the second page is loaded; the packets of the second page overlap with that of the first. We discard the *overlapping chunk* entirely and classify web pages using the *initial chunk* of packets, i.e., packets before the second pages. In order to achieve this goal, the attacker must correctly identify the split point by utilizing a split finding algorithm to obtain the chunk and classify pages with the restricted chunk of packets.

To develop a successful WF attack with relaxing the Single Page Assumption, we make the following novel contributions:

1. We propose a multi-tab fingerprinting attack that allows an attacker to classify web pages with multi-tabs, which is not well addressed by the literature.
2. We present a new BalanceCascade-XGBoost algorithm to accurately identify the split point, given a combination of two pages.
3. We develop a new classifier based on random forests, which accurately classifies web pages given only the initial chunk of packets according to the selected features.
4. Experimentally, we verified the success of our new WF attack in a multitude of scenarios, including datasets collected with SSH, Tor, and under several recently proposed defenses against WF. We found that our new WF attack can achieve 93.88% True Positive Rate (TPR) on two seconds of the initial chunk against SSH-loaded data. It can also achieve 77.08% TPR on six seconds of the initial chunk against Tor-loaded data, which beats the previous best attack, i.e., *k*-FP [11], by more than 9%.

The rest of the paper is organized as follows. In Section 2, we present the threat model and introduce related work. We present the framework of our new WF attack in Section 3. We present our BalanceCascade-XGBoost algorithm to split pages in Section 4 and classify pages in Section 5. We evaluate the effectiveness of our attack in Section 6. Section 7 concludes the paper.

2 PROBLEM STATEMENT AND RELATED WORK

2.1 Multi-tab Threat Model

In the WF threat model, the adversary records the encrypted network flow between the victim and the proxy. To determine whether the encrypted network flow is generated by a targeted page, the adversary constructs his fingerprint database by extracting various network flow features of the targeted page, such as the directions and sizes of packets. The adversary eavesdrops on the victim's network flow and classifies the victim's network flow using a supervised classifier trained on his fingerprint database.

When the victim opens multiple tabs simultaneously, the browser generates overlapped network flows corresponding to different pages through the same connection. The attacker cannot distinguish between overlapped network flows [14, 27]. To avoid this issue, WF attacks are evaluated with the Single Page Assumption: only one page is visited at a time and no background network flow is generated. The assumption is unrealistic. Network flows generated by the same client are always overlapped [14, 23, 27]. Juarez et al. showed that current WF attacks fail against overlapped network flows [14].

In this work, we address the Single Page Assumption and extend the threat model. In the extended threat model, the client visits a page, waits for a short period (referred to as the delay), and opens another page in a new tab, which is called sequential multi-tabs (for short, multi-tabs). If the first page does not finish loading before the time gap, the two pages will be loaded simultaneously, and their network flows will overlap. With such sequential multi-tab pages, only the initial chunk is clear and can be used to launch the fingerprinting attack. Therefore, the goal of our attack is to fingerprint a website by accurately identifying the first page obtained from the website. For easy illustration, this paper focuses on the two-tab scenario. In Section 6.5, we will illustrate that our attack is still effective if there are more than two sequential pages.

2.2 Related work

Single Page Website Fingerprinting Attack Single page website fingerprinting attacks use the whole network flow to identify web pages visited by clients [2, 5, 6, 9, 12, 13, 16, 18, 22, 24]. In 2014, based on more than 3000 features extracted from network flows, Wang et al. [25] presented a k-Nearest Neighbours (kNN) classifier with weight adjustment, which achieves TPR of 0.85 and FPR of 0.006 on Tor. In 2016, Panchenko et al. [21] presented a new approach, CUMUL, which uses SVM with only 104 features; they showed that CUMUL achieves better results than kNN. Hayes et al. [11] created a K-FP attack that utilizes random forests to extract fingerprints for each network flow and then train a kNN classifier by the fingerprints. This attack shows better results under defenses compared with Wang's kNN attack and Panchenko's CUMUL attack. Unfortunately, these attacks cannot effectively identify pages if there exist multiple tabs.

Multi-tab Website Fingerprinting Attack Juarez et al [14] showed that known WF attacks fail without the Single Page Assumption: they cannot identify two pages that are loaded simultaneously. There are two major works that have attempted to address this issue. Gu et al. [10] relaxed the assumption about browsing behavior and presented a WF attack on the multi-tab scenario. Using the same extended threat model as ours, they selected fine-grained features such as packet order to identify the first page and utilized coarse features to identify the second page. With a delay of two seconds, when accessing the top 50 websites using SSH, according to Alexa, their attack can classify the first page with 75.9% TPR, and the second page with 40.5% TPR in the closed-world setting where all the pages are monitored. Our attack achieves a higher fingerprinting accuracy by finding accurate split points of the second pages. Even with the same split points, our attack shows a much higher TPR on the first page.

The work of Wang and Goldberg [27] is most closely related to our approach. They attempted to separate network flows using either a noticeable time gap or their split finding algorithm, i.e., time-based KNN (time-kNN). Then, they classified split pages using the kNN attack from 2014 [25]. The effectiveness of the attack is limited whenever two pages were loaded simultaneously. We will show a superior split finding algorithm using BalanceCascade on

XGBoost as well as a WF classifier better suited for classifying a small chunk of packets, which ensures that our attack achieves the accuracy of multi-tab fingerprinting.

3 OVERVIEW OF MULTI-TAB ATTACKS

In this section, we present a new WF attack that effectively fingerprints web pages opened with multiple tabs. Our attack aims to relax the Single Page Assumption used in the existing WF attacks. It aims to accurately identify pages with multiple tabs by classifying the pages only with the initial chunks. In order to achieve this, we develop two classifiers, the first one is used to identify the split points of the pages and the second one is to classify the pages according to the initial chunks after page split.

The key observation behind our attack is that, if clients want to open pages with multi-tab pages, they normally open the second page after some delay, i.e., sequential multi-tab pages. For example, a client may spends reading some contents of the first page before selecting a link to a new page [23]. Thus, our attack should be able to always identify pages visited by clients as long as it can successfully identify the split points of the pages and obtain the first chunks for page classification. However, it is challenging to achieve the goal. In particular, in order to construct the multi-tab attack, we should answer the following two questions in this paper.

- Is it possible to accurately identify split points of different pages? In particular, the pages are opened by various clients with arbitrarily delay.
- Is it possible to classify the first chunks after the split such that an attacker can accurately identify the pages? Specially, the first chunks are with a small number of packets.

Note that, in theory, we could identify web pages with any numbers of tabs as long as we can successfully obtain the first chunks of the pages. For simplicity, in this paper, we only consider the attack with two-tab pages to demonstrate the feasibility of the attack.

4 DYNAMIC PAGE SPLIT

In this section, we present our page finding algorithm that allows an attacker to accurately understand when the second page starts so that the attacker can identify the page with the initial chunk.

4.1 Challenges in Identifying True Split Points

We extract 23 features according to the study of Wang and Goldberg [27] to identify split points of the pages. As mentioned above, the split point we want to find is the start point of the second page, which we refer to as the "true split". It allows us to eliminate the noise of the second page and obtain all the non-overlapped part of the first page. In the web browsing process, a client sends an outgoing packet to request web page resources from the server, which means that the start point of the second page can be any outgoing packet.

However, loading a web page may trigger multiple outgoing packets, one of which is "true split" of the pages. It is difficult to find the "true split". In particular, the number of outgoing packets is large. In order to correctly identify the second pages, in the training phase, we should place the true split in the "true splits" class and all other outgoing packets in the "false splits" class. Thus, there are only one "true split" and multiple "false splits" in the analyzed network flow instance, which incurs an unbalanced classification issue. According to our study with real datasets, we find that the proportion of positive and negative instances can reach 1:461. The goal of most existing learning algorithms is to reduce the overall classification error. In these algorithms, all instances are treated equally and the error of the different classes of misclassification is the same. Considering the ratio of the number of the "true splits" class to that the "false splits" class in our datasets, even though all the instances are predicted as the "false splits" class and the accuracy of our classifier can reach 99.78%, we cannot accurately identify most "true splits". The unbalanced training set will lead to the result that the classifier classifies the "false splits" class with high classification accuracy and the "true splits" class with low classification accuracy. Thereby, it is challenging to use a classifier to find the start point of the second page.

4.2 BalanceCascade-XGBoost Algorithm

To address this issue, we propose our split finding algorithm, i.e., BalanceCascade-XGBoost, which is an undersampling method combining the BalanceCascade method [17] and the XGBoost classifier [7] to train a binary classifier. In the testing phase, the classifier calculates the individual probability of every outgoing packet belonging to the "true splits", and then classifier guesses the most probable outgoing packet that may be the "true splits". We randomly obtain multiple b "false split" class instances and one "true splits" class instance from each network flow.

Given our training dataset D, where the ratio of the number of classes in "false splits" class N to the number of instances in "true splits" class P is b:1. In our BalanceCascade-XGBoost algorithm, each time we randomly select N_i from N, where $|N_i| = |P|$ (i is the round of sampling), and then compose a training subset D_i by using N_i and P. Then we train a kNN classifier [8][1] with default parameter k=1 using training subset D_i, and then remove the instances in N that are correctly classified by kNN classifier. We continue to sample another training subset D_j from D. In the end, with the help of BalanceCascade, we have a collection of $D_i, i = 1...n$ training subsets, $D_i = \{(x_j, y_j)\}(|D_i| = 2|P|, x_j \in R^m, y_j \in \{0, 1\})$, where x_j is a feature vector extracted from candidate split points, $y_j = 0$ is the "false splits" class, 1 for the "true splits" class, and m is the dimension of the feature vector.

Moreover, we utilize the XGBoost classifier [7] in our BalanceCascade-XGBoost algorithm to boost trees on a large amount of data. XGBoost is a massive parallel boosted tree tool, which is a widely used boosted tree tool and achieves more than ten times faster than the other popular classifiers. We train an XGBoost classifier with each training subset D_i. The hypothesis function of XGBoost is an ensemble of regression trees, where regression tree [4] is a tree whose leaf node stores a class value that represents the average value of each leaf node's instances. When we solve our binary-class task, the ensemble of regression trees outputs a real number value and XGBoost uses the Sigmoid function to convert the output to be a value close to 0 or 1, where 0 is the probability of being "false splits" class, and 1 is the probability of being "true split" class. In

[1]Our algorithm uses the kNN classifier instead of the AdaBoost classifier used in the original BalanceCascade algorithm since the kNN classifier can achieve better performance according to our study.

the construction of XGBoost, regression trees are constructed one by one incrementally, and thus it is impossible for XGBoost to construct each tree in parallel. Fortunately, the training data can be sorted in advance before training so that it can be organized with a block structure. This block structure makes parallelism possible. During the splitting of a node, the gain of each feature will be recalculated, and the gain calculation of each feature can be performed by multiple threads.

We use each D_i training subset to train a weak XGBoost classifier f_i, and then combine all the weak classifier to compose a final classifier $F(x)$ "ensemble of XGBoost", which actually is an "ensemble of forests of regression trees". Here we have n training subsets. The hypothesis function of our final classifier can be computed as follows.

$$F(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x). \tag{1}$$

To find the true split, the classifier tests every outgoing packet in the network flow and then outputs the probability of each candidate split point. Finally, the classifier returns the outgoing packet that has the highest probability of being the true split.

5 CHUNK-BASED PAGE CLASSIFICATION

In this section, we develop a classifier to classify the initial chunks of pages obtained by page split. In particular, it utilizes a random forest ensemble classifier, based on extensive feature sets which are first trimmed down using feature selection.

5.1 Feature Selection

Now we present the feature set selected by our attack. Since we only use the initial chunk to extract features, many features used in the previous work cannot be applied, e.g., total transmission time. Inspired by the prior WF work [11, 21, 25], we choose 452 candidate features, and develop a feature selection algorithm to select the most useful feature subset based on the training subset. We utilize IWSSembeddedNB [1], an incremental wrapper subset selection embedded Naïve Bayes classifier [20], to select the most useful features in our attack.

First, we rank all the features in descending order using symmetrical uncertainty (SU), which is used to compute the correlation between individual features and classes by the following equation:

$$SU(f, y) = 2\left(\frac{SE(f) - SE(f|y)}{SE(f) + SE(y)}\right), \tag{2}$$

where y indicates a class and $SE(f)$ is the Shannon entropy for each feature f. Our feature subset is S; we try to add the feature into S from rank list, starting from the first one.

Second, we use a Naïve Bayes classifier to compare the true positive rate (TPR) between $S \cup f$ and S over the training subset, and add f into S only if the TPR increases with f. Finally, we can obtain the feature set S. The most representative features selected by our algorithm are listed as follows, and the rest can be found in Appendix A.

- The round trip time (RTT), which is the delay between the first outgoing packet and the first incoming packet.
- Document length and the size of incoming packets rounded to the nearest multiple of 100.
- The total size of incoming packets and total packets, the ratio of the total size of incoming and that of outgoing packets to the total size of the network flows.
- The number of outgoing packets, and the fraction of the number of outgoing packets and that of incoming packets in the first 20 packets of the network flows.
- Statistics of packet ordering. We extract the total number of packets before the next incoming or outgoing packet recorded in the network flow, obtain two lists for incoming and outgoing packets individually, and then compute the standard deviation and the average deviation of the two lists.
- Burst sizes and quantity. In Wang et al.'s kNN attack [25], Wang et al. defined burst as a sequence of outgoing packets, which is triggered by one incoming packet. We sample 20 bursts. We select the size sequence of 20 bursts as the bursts' size features (BSF) and the quantity sequence of 20 bursts as the bursts' quantity features (BQF). Note that, the 2-4th BQF and the 1-5th BSF are included in the feature subset.
- The cumulative size of packets (CSOP) [21]. We sample 100 CSOP features as recommended by[21]. Note that, the 2-6th, 8-11th, 29th, 98-99th CSOP features are selected in the feature subset.
- Statistics of transmission time. We extract all three quartiles from the total, incoming and outgoing packet sequences, and extract the total transmission time from the incoming and outgoing packet sequences. Note that, only the first quartile of the total, incoming and outgoing packet sequences and the second quartile of the total packet sequence are selected in the feature subset.
- Statistics of packet inter-arrival time. We extract three lists of inter-arrival times between two packets of the network flow for total packets, incoming packets, and outgoing packets. We collect the statistics: max, mean, standard deviation, and the third quartile features from each list. Note that, maximum inter-arrival time of total packets and incoming packets, and minimum inter-arrival time of incoming packets are selected by feature selection and included in the feature subset.
- Fast Levenshtein-like distance (FLLD) [26], where each class has a FLLD feature measuring the similarity of two instances by its packet size and order. Note that, the 1-4th, 6th, 8th, 12th, 16-17th, 19-23th, 26th, 28-31th, 33-35th, 39-40th, 42th, 44th, 46-50th FLLD features are selected in the feature subset.
- Jacquard similarity with unique packet length (JSWUPL) [16], measuring a Jacquard similarity between each instance and the total instances associated with each website with respect to the unique packet length of each page. Note that, the 16th, 18th, 31th, 35th JSWUPL features are selected in the feature subset.

5.2 Classifier Design

Our classifier is built upon random forests [15], which identifies the first page based on the features we selected. On each training set D_i, where $|D_i| = m$ for each $1 \le i \le n$. n is the number of decision trees, which we set to 100. We select n data subsets and uniformly sample m packet sequences with replacement of a total of m times to obtain n subsets of i.i.d.

Random forests are ensemble classifiers which consist of a collection of weak classifiers $h_i(x)$. The weak classifiers $h_i(x)$ are random forest decision trees [4] that use the Gini index to grow the tree. Each x is an input feature vector we extracted from the network flow; a decision tree classifier is trained by $x \in D_i$.

The training process of the random forest decision tree selects a random subset I which contains k features from the feature set d of the node, and then the tree chooses the best feature from I to grow the tree, where k is recommended as $\log_2 d$ in [3]. Note that, the training process is different from the traditional CART decision tree. The traditional CART decision tree selects the best feature using the Gini index from all the features belonging to the node to grow the tree, which cannot achieve the diversity of classifiers.

Here, we assume we have P classes labeled as $\{c_1, c_2, c_3 ... c_P\}$. Given a testing element for classification, each $h_i(x)$ separately classifies the element and outputs a label vector of P dimensions , i.e., $[h_i^1(x), h_i^2(x), \ldots, h_i^P(x)]$, where $h_i^j(x)$ indicates the output of $h_i(x)$ on label c_j, and then random forests classifier $H(x)$ labels the input x with the most popular class. Thus, our random forests classifier can be expressed as follows:

$$H(x) = c_{\arg\max_j \sum_{i=1}^{n} h_i^j(x)}. \tag{3}$$

6 EXPERIMENTAL RESULTS

In this section, we describe the collected datasets used in our experiments in this paper, and then present our experimental results.

6.1 Experiment Setup

Single-Tab Datasets: We collect three datasets: SSH_normal, SSH_noisy, and Tor_normal. In each instance, the network flow corresponds to one page. We choose to monitor the web pages from Alexa's top-ranked websites[2]; Alexa is a website collecting the most visited URLs, which is widely used in previous WF studies. SSH_normal consists of 50 monitored web pages over SSH with 50 training instances and 50 testing instances for each page. There are a total of 100 instances for each page without any background network flow. SSH_normal also contains 2500 unmonitored web pages chosen from Alexa's top 5,000 websites. We collected the SSH_normal dataset with a headless browser, PhantomJS[3], and we used tcpdump to record the network traces. Similar to the work in the literature [26], pages are retrieved without caching, and we wait for two seconds after a page finishes loading before fetching the next one.

Moreover, in order to verify the effectiveness of our proposed attack in the real world, we collect a SSH_noisy dataset. The difference between SSH_noisy and SSH_normal is that the web pages in SSH_noisy contain dynamic content such as audio and video. SSH_noisy is generated by accessing 50 chosen web pages without any background network flow, including 50 training instances and 50 testing instances for each page. As PhantomJS cannot load dynamic content, we used Selenium[4] for SSH_noisy.

Tor_normal is collected by automatically visiting pages using Tor Browser 6.5.1[5]. It includes the same pages and number of instances to SSH_normal. Tor_normal consists of three subsets of web pages: (i) 50 instances from each of 50 monitored web pages without background noise as training subset, (ii) another 50 instances from each of 50 monitored web pages without background noise as testing subset, (iii) a open world dataset that included the instances with 2,500 unmonitored web pages.

Two-tab Datasets: We collect two datasets, SSH_two and Tor_two, where each instance contains two pages instead of one. In each instance, we access two chosen pages, and the second page that is randomly selected from the monitored pages is loaded with a time gap. Since delays of most page retrieval are larger than two seconds [23], we set the minimal initial chunk size to be two seconds. In addition, according to our observation (see Section 6.4), we find six seconds are enough to collect packets for the attacks and thus we set the maximum initial chunk size to six seconds. Therefore, in our experiments, we collect the SSH_two dataset with five different time gaps: two, three, four, five, and six seconds. For each time gap, we collect 50 monitored web pages, each with 50 instances. We collected Tor_two with random time gaps. We access two pages at the same time, and the second page is opened with a random time gap. We collected 5000 instances, which is used for our dynamic split experiments. The time gap ranges from two seconds and six seconds with the same setting as SSH_two. With each time gap, we also have 50 web pages with 50 instances for each page.

Data Preprocessing. The essence of the WF attack is a page classification problem, and the effectiveness of the attack is affected by noise. Thus, we need to preprocess the data to remove the noise. In the SSH dataset, if a flow has fewer than 20 packets before the second page loads, we treat it as failed page loading and throw away the instance. In addition, we throw away packets with the lengths of 100, 44, 52, or 36, because these are likely to be SSH control packets. We also throw away TCP ACK packets whose lengths are 0. We note that an attacker can do the same to improve the effectiveness of the attack and thereby our preprocessing does not enhance the attacker's power. As for the Tor dataset, we throw away instances with fewer than 75 Tor cells. Note that, we observe that, in Tor_two, the second pages are loaded with a random delay after the first pages. In order to accurately evaluate the effectiveness of our attack, we use the Tor_normal testing subset to generate a new dataset with two pages called Tor_twop, where the second page is randomly selected from the Tor_normal testing subset with the delay of one-second granularity. Actually, if we use a finer granularity, we could obtain a higher TPR with more collected packets (see Section 6.4). Similarly, we also use the Tor_normal training subset to create Tor_split, where two pages are loaded with the delay of one-second granularity, which is used to train our classifiers to identify different pages in Tor. We create 50 monitored web pages in Tor_split for each time gap and each web page include 50 instances.

Metrics. In this paper, we use false positive rate (*FPR*) and true positive rate (*TPR*) to measure the effectiveness of our attack. FPR measures how often unmonitored instances are wrongly classified as monitored ones and TPR measures how often monitored instances are correctly classified. Note that, for simplicity, except split evaluation experiments, we use split times to measure the chunk sizes. Split times are trimmed down from the original split points so that they are with one-second granularity.

[2]http://www.alexa.cn/

[3]PhantomJS is a headless WebKit scriptable with JavaScript: http://phantomjs.org/.

[4]Selenium is a suite of tools to automate Chrome and Firefox web browsers across various platforms: https://www.seleniumhq.org/.

[5]https://www.torproject.org/projects/torbrowser.html.en

Table 1: Detection accuracy with respect to specified split time with our new WF attack on SSH. For each real split time, the table shows the TPR of network flows detected as each specified time. Bolded values represent TPR when specified split time is equal to real split time.

		Real split				
		2s	3s	4s	5s	6s
Specified split	2s	**94.2%**	93.56%	93.68%	93.76%	93.2%
	3s	70.08%	**93.68%**	92.44%	92.84%	93.56%
	4s	56.68%	73.96%	**92.4%**	91.4%	92.16%
	5s	54.08%	59.76%	77.48%	**92.04%**	92.32%
	6s	39.72%	44.24%	55.16%	76.16%	**90.6%**

6.2 Evaluation of Multi-tab] Attacks

In this section, we evaluate our WF attack based on the dynamic split on both SSH and Tor.

Evaluation on SSH. In this experiment, we train classifiers with split times set to two, three, four, five, and six seconds using the training subset of SSH_normal. In the testing phase, we test on SSH by loading two pages with a different delay time, where we extract the features from the initial chunk of network flow with specified split time. We show the results in Table 1. When the split time is correctly detected, the TPR values are shown in bold. We observe that, if the split time later than the real-time, it incurs worse results than when the split time is earlier than the real time, which can be seen in each column of the table. Surprisingly, we find that setting the split time to be two seconds (see the first row of the table) is slightly better than dynamic page split by around 3%. However, our dynamic split ensures our attack can succeed in various scenarios, e.g., on both SSH and Tor.

Evaluation on Tor. In this experiment, we use the same setting as that in our SSH experiments. We train classifiers with split times between two seconds and six seconds, and test with instances in Tor_twop. Table 2 illustrates the results of detected delay with respect to various specified split time. Different from the SSH results, we find the dynamic split is necessary for Tor. The bold TPR values with the corrected detected split time are the best in each column. The difference between the results on SSH and Tor is probably due to the fact that most pages loaded with several seconds on SSH but with much longer time on Tor. It demonstrates that our dynamic split finding techniques (BalanceCascade-XGBoost) are necessary for Tor. Note that, the TPR is constant for each row when the specified split time is lower than the real split time because our dataset for Tor_twop was synthesized by combining packet sequences within a time gap.

Table 3 shows that the detection results with respect to various real split time. We train classifiers using Tor_normal with varied split time between two seconds and six seconds, and use Tor_split to train a BalanceCascade-XGBoost classifier with the same features as Section 6.3. We see that the error rate increases when the split time decreases. The reason is that many features cannot be extracted within shorter split time. The larger expected round-trip time on Tor, fewer packets in smaller split time. We also can observe that the classifier appears biased in the classification with larger split time. Almost all incorrect detected split times are with the real split time of six seconds because we set the maximum delay time to be six seconds. As shown in Figure 2, dynamic split WF attack performs much better than that with any other specified split time on Tor,

Table 2: Detection accuracy with respect to various specified split time with our new WF attack on Tor. For each real split time, the table shows the TPR of network flows detected at each specified time. Bolded values represent TPR when specified split time is equal to real split time.

		Real split				
		2s	3s	4s	5s	6s
Specified split	2s	**50.4%**	50.4%	50.4%	50.4%	50.4%
	3s	47.2%	**64.76%**	64.76%	64.76%	64.76%
	4s	44.28%	59.4%	**68.76%**	68.76%	68.76%
	5s	42.6%	59.64%	67.32%	**73.36%**	73.36%
	6s	41.6%	57.32%	67.28%	71.32%	**77.08%**

Table 3: How often BalanceCascade-XGBoost outputs each possible split time for each real split time. Bolded values represent correct detection of split time. There are 2500 instances for each split time.

		Real split				
		2s	3s	4s	5s	6s
Detected split	2s	**51.96%**	0.72%	0.64%	0.44%	1.04%
	3s	4%	**66.48%**	1.28%	0.92%	1.52%
	4s	4.72%	3.24%	**65.72%**	1.4%	1.76%
	5s	6.2%	5.28%	5%	**74.28%**	3.04%
	6s	33.12%	24.28%	27.36%	22.96%	**92.64%**

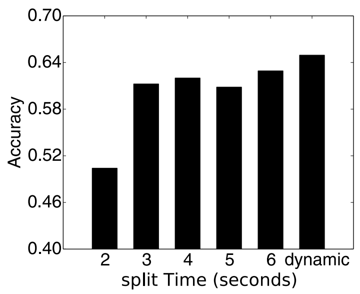

Figure 2: TPR of detected split time with our dynamic split WF attack on Tor that loads two pages with the different time gap, comparing with the assumed split time ranging from two seconds to six seconds.

which achieves a TPR of 64.94%. It is interesting to see that the TPR with different specified split time is random, which means it is hard to select and set a fixed split time to construct the attack. However, our dynamic split WF attack solves the problem and achieves a higher TPR than any other specified split time. It is not surprising to see that, compared with CUMUL and k-FP, our attack achieves the best attack accuracy (see Figure 3).

Note that, in our experiments, we use one-second granularity to set various split times. In practice, an attacker can use finer granularity, e.g., 500 milliseconds, when training classifiers, which may achieve a higher TPR.

Attack Against Defenses. We observe that feature selection was especially useful to defeat some defenses proposed in the WF literature. The reason is that WF defenses significantly change the shape and characteristics of the client's traffic. We tested the following defenses, and evaluated defenses when applied to the SSH_normal[6].

[6]We used Wang's code at https://cs.uwaterloo.ca/ t55wang/wf.html to create defenses.

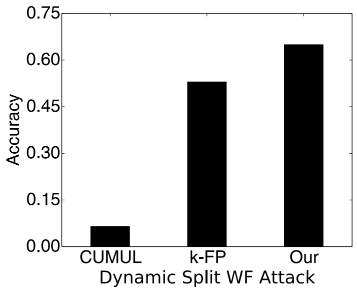

Figure 3: TPR of detected split time with our dynamic split WF attack on Tor that loads two pages with the different time gap, comparing with CUMUL and k-FP.

Table 4: Comparison of the TPR of our classifier with CUMUL and k-FP against various defences with an initial chunk size of two seconds.

Defense	CUMUL	k-FP	Our
HTTPOS split	2.16%	88.12%	93.92%
Traffic morphing	2.96%	80.08%	84.52%
Decoy pages	2.92%	70.36%	74.16%
BuFLO	4.4%	12.6%	12.32%

- Traffic morphing [28], which alters the packet sizes of the client's traffic according to the packet distribution of a target web page, used as a decoy for the real web page.
- HTTPOS split [19], which utilizes HTTP range requests to obfuscate the size of small outgoing and incoming packets, splitting them into random sizes.
- Decoy pages [22], which loads a decoy page whenever the client opens a new web page.
- BuFLO [9], which sends packets at a constant size and at regular intervals in both directions.

We compare our classifier with the state of the art, i.e., k-FP and CUMUL, on various defenses in the closed-world setting. Our defense datasets are converted from SSH_normal. Table 4 shows the performance of all three classifiers under various defenses, where the initial chunk size is two seconds. Against each defense, our attack is comparable to or performs better than both k-FP and CUMUL. Surprisingly, against HTTPOS split, we achieve almost the same TPR as on the SSH_normal dataset, which means the HTTPOS split has no influence on TPR. When traffic morphing is applied, our attack can achieve 84.52% TPR, which is better than K-FP by more than 4%. It is interesting to find that our attack achieves 74.16% TPR when decoy pages are used. Usually, in decoy page defense, we load another page at the same time when we open a target page. In theory, background noise and network flows of the target page are mixed and there is no non-overlapped part. However, as long as there is delay between the load time of these two types of pages, we can still classify the target page. Here, we use unmonitored pages as the noise [25]. Our new WF attack can still achieve high TPR with the initial chunk against defenses even when the split time is only two seconds. We observe similar TPR if the initial chunk size is larger than two seconds.

6.3 Evaluation of Page Split

Now we compare the performance of BalanceCascade-XGBoost with time-kNN [27] that achieves page splitting. If the start point

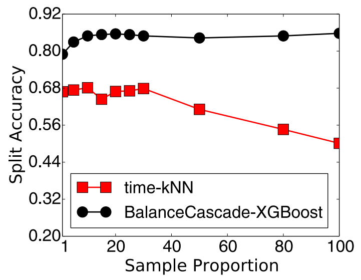

Figure 4: The split accuracy of BalanceCascade-XGBoost compared to time-kNN while varying the proportion of "false splits" class to "true splits" class.

of the second pages is within 50 packets of the first page, time-kNN cannot classify such pages. To perform a fair comparison between the two algorithms, we filter the instances which do not satisfy the requirement of time-kNN. We use the same features used in time-kNN to compare the performance of the two algorithms. We generate the SSH_random and Tor_random datasets that are randomly selected from SSH_two and Tor_two datasets with two pages, respectively. We train 1,500 instances in each dataset and then use 1500 instances for test.

Accuracy Evaluation with Varying Sampling. As we discussed above, the binary classification has a problem that there is a serious unbalance between "true splits" and "false splits" classes. The BalanceCascade used in our split finding algorithm resolves this issue by balancing the quantity of two classes in an under-sampling method. We evaluate how the proportion b of "false splits" and "true splits" classes affect our split accuracy compared with time-kNN. Here, the split accuracy is defined as the percentage of packet sequences of the dataset on which our algorithm returns a split point that is fewer than 25 packets before or after the true split point.

As shown in Figure 4, we can see that, even when the ratio of the number of "false splits" class to that of "true splits" class is 1:1, our split accuracy is higher than time-kNN. However, the accuracy is relatively stable after b is larger than 10. On the one hand, more "false splits" class instances introduce more information about the false split point. On the other hand, more "false splits" class instances mean more weak classifiers built and thus incur more computation time. Hence, we consider $b = 10$ is a good choice. However, the split accuracy of time-kNN decreases with b increases, which means time-kNN has a limitation in an unbalanced dataset. Thus, time-kNN achieves low classification accuracy if the classes are unbalanced.

Next we compare the split accuracy of our algorithm with time-kNN. In this experiment, the ratio of the number of "false splits" class N to the number of "true splits" class P is 10:1 in our training dataset. Figure 5 shows the performance of our algorithm compared to time-kNN algorithm. On the Tor dataset, we have achieved a higher split accuracy than time-kNN. They achieve the split accuracy of 82% and 69%, respectively. Note that, the split accuracy of guessing correct split randomly for any outgoing packets is only 0.22% with the Tor_two dataset. As for SSH dataset, when an SSH

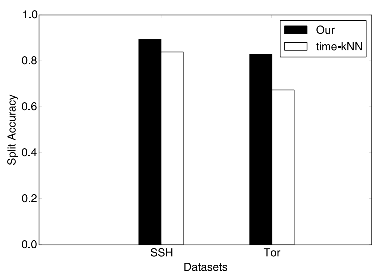

Figure 5: The split accuracy of BalanceCascade-XGBoost compared to time-kNN on SSH and Tor dataset.

Table 5: TPR of split time detection with time-kNN on SSH. With each real split time, the table shows the number of network flows detected as each possible split time, where bolded values represent correctly detected split time.

		Real split				
		2s	3s	4s	5s	6s
Detected split	2s	**961**	44	28	29	40
	3s	46	**932**	20	24	21
	4s	34	77	**1024**	32	30
	5s	21	40	44	**1028**	21
	6s	28	41	67	112	**1095**
	N/A	160	116	67	25	43

client requests to open a new channel, it will issue an SSH-MSG-CHANNEL-OPEN message of 100 bytes to SSH server. Because each new page will ask for an SSH-MSG-CHANNEL-OPEN message, we reduce the number of candidate points by abandoning these candidates whose length is not 100, which helps to improve the split accuracy of SSH dataset. According to Figure 5, the split accuracy of SSH dataset is much better than that of Tor. We can detect the "true split" with an 89.41% split accuracy. Therefore, our algorithm well outperforms time-kNN.

Splitting Time Evaluation. In this experiment, we use the detected split time to evaluate the performance of our BalanceCascade-XGBoost against time-kNN. We use the Tor_twop and SSH_two to do the experiments, respectively. For instance, we use the half of the Tor_twop dataset to train the classifier. In each subset of different delay time, each web page has 25 instances in the training phase, and then we use the remaining to test. Due to that some true split point is near the start of network flow, we want to detect all the outgoing packets. We extract the same features as time-kNN except for the features of mean, standard deviation, and maximum inter-cell time for twenty cells before and after the candidate cell. We measure all the outgoing packets, and set 0 to the feature by default if we cannot get enough packets to extract the corresponding feature.

As we described above, time-kNN has ignored some instances due to their start point of the second page is within 50 packets of network flow. According to Table 5 and 6, each column has 1250 instances. N/A in the table means the number of instances that cannot be captured by time-kNN. In each split time, BalanceCascade-XGBoost has more true positive instances than time-kNN on SSH. According to the columns shown in Table 6, the number of detected later is larger than the number of detected earlier, which means it

Table 6: TPR of split finding with BalanceCascade-XGBoost on SSH. For each real split time, the table shows the number of network flows detected as each possible split time. Bolded values represent correctly detected split times.

		Real split				
		2s	3s	4s	5s	6s
Detected split	2s	**1086**	25	14	18	21
	3s	51	**1112**	27	11	15
	4s	47	52	**1112**	22	14
	5s	45	28	61	**1148**	25
	6s	21	23	36	51	**1175**

Table 7: TPR of split finding with time-kNN on Tor. For each real split time, the table shows the number of network flows detected as each possible split time. Bolded values represent correctly detected split times.

		Real split				
		2s	3s	4s	5s	6s
Detected split	2s	**124**	15	24	25	19
	3s	13	**252**	29	31	32
	4s	29	39	**388**	56	56
	5s	35	64	69	**590**	74
	6s	60	91	167	164	**787**
	N/A	989	789	573	384	282

Table 8: TPR of split finding with BalanceCascade-XGBoost on Tor. For each real split time, the table shows the number of network flows detected as each possible split time. Bolded values represent correctly detected split times.

		Real split				
		2s	3s	4s	5s	6s
Detected split	2s	**644**	8	9	9	16
	3s	46	**818**	15	16	17
	4s	65	43	**822**	23	26
	5s	71	60	60	**910**	36
	6s	424	321	344	292	**1155**

is much easier to detect later than earlier when the detection of split finding is wrong. With real split time increase, we can see the decrease of the number of N/A as well as the increase of corrected instances, which means longer delay time is advantageous to split finding. As shown in Table 6 and 8, we find it is more difficult to identify web pages on Tor. In addition, Table 7 and 8 illustrates that the number of our true positive of all the split time is larger than time-kNN. In a nutshell, we can conclude that our Balance-XGBoost is better than time-kNN in both SSH and Tor scenario.

6.4 Evaluation of Chunk-Based Classification

In this section, we evaluate the performance of our new WF classifier. Therefore, the attacker trains and classifies with only the initial chunk, i.e., the packets before the split point (corresponding to one page). We perform experiments under the closed-world and open-world settings. In the closed-world setting, we test our classifier with a dataset where all web pages are monitored, while in the open-world setting, the dataset consists of both monitored and unmonitored web pages. In order to fairly compare with previous classifiers, we will use a short initial chunk to compare these existing classifiers, which makes our classifier the optimal choice for the multi-tab scenario.

datasets	Before TPR	After Feature Selection TPR	After Feature Selection # of Features
SSH_normal (2s)	93.96%	95.84%	79
Tor_normal (2s)	56.64%	56.2%	31
Tor_normal (3s)	68.04%	68.72%	48
HTTPOS split (2s)	91.48%	93.6%	81
Traffic morphing (2s)	78.16%	83.64%	82
Decoy pages (2s)	75.36%	80.06%	92
BuFLO (2s)	13.08%	13.2%	15

Table 9: Comparison of TPR before and after feature selection. The number of features before feature selection is 452, where 2s and 3s indicate the split time of two seconds and three seconds, respectively.

Results of Feature Selection. In the experiment, we evaluate the TPR of our new WF attack after performing feature selection on each dataset using IWSSembeddedNB, while limiting each packet sequence to two seconds of the initial chunk. We calculate the TPR using ten-fold cross-validation in the closed-world setting. The datasets we use include both the SSH_normal and Tor_normal datasets. We also do experiments on Tor_normal when the split time is three seconds.

According to Table 9, we can see that our attack achieves higher TPR after feature selection (except on the Tor_normal training subset when the split time is two seconds), even though we are using strictly less information after feature selection. The feature subset each dataset uses is described in Appendix B. In the case of SSH_normal, the jump in accuracy is especially surprising: an almost two-percent increase in the true positive rate corresponds to a one-third decrease in the false negative rate (6% to 4%). Our final TPR of 95.84% compares favorably with state-of-the-art attacks, even though we only used two seconds of data. Against BuFLO, feature selection left us with only 15 features that relate to time statistics, as size and ordering features are removed by BuFLO. As for Tor_normal, we are interested to find that with feature selection we only utilize 31 features, and we can achieve a similar TPR as 452 features in the Tor_normal dataset when the split time is two seconds, even more, we have a slightly higher TPR with fewer features when the split time is three seconds.

Attack Under the Closed-world Scenario. We compare our classifier with k-FP [11] (which also uses random forests) and CUMUL [21] on three datasets above: SSH_normal, SSH_noisy, and Tor_normal. For each dataset, we use the training subset to train the classifiers and the testing subset to test. In the closed-world setting, we set the number of features k in the random subset I to 100. The reason why we have such setting is that we do not consider feature selection here such that we can systematically study the performance of our attack without selection and our new WF attack can achieve the highest TPR when k is set to 100 according to our studies. For k-FP, we set the parameters according to Hayes et al.'s code[7], and for CUMUL, we scale each feature linearly to the range of [-1,1] and search from the range of parameters recommended by [21]. Figure 6, 7, and 8 show that TPR with the SSH_normal, SSH_noisy, and Tor_normal datasets, respectively. According to the experiment results, we made the following observations:

[7]https://github.com/jhayes14/k-FP

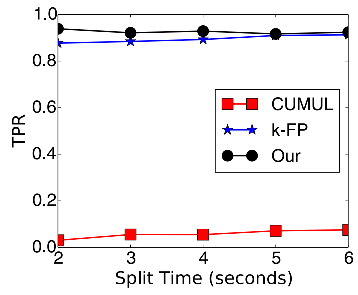

Figure 6: TPR of our classifier compared to k-FP and CUMUL while varying the initial chunk size between two seconds and six seconds on the SSH_normal dataset.

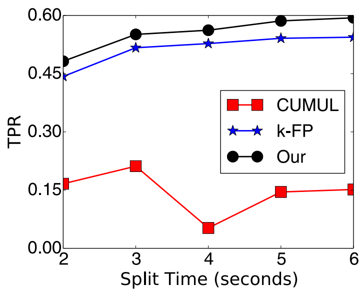

Figure 7: TPR of our classifier compared to k-FP and CUMUL while varying the initial chunk size between two seconds and six seconds on the SSH_noisy dataset.

- **Our classifier achieves better TPR on the initial chunk.** Our classifier outperforms k-FP though they have similar fine-grained features such as the number of packets and packet ordering. This may be because we also use high-level features, e.g., Fast Liechtenstein-like distance, which can be used to describe the correlation between two network flows in a coarse manner. CUMUL always had a low TPR even though we tried various parameters. According to the previous studies [11], CUMUL does not outperform k-FP, which is consistent with our results.
- **Increasing the initial chunk size does not always increase TPR.** According to Table 6, on SSH_normal, the TPR of our new WF classifier slightly decreases when the split time increases due to the limited number of instances. On SSH_noisy and Tor_normal, the TPR increases when the split time increases. In particular, on Tor_normal, our classifier achieves a TPR of 50.4% for an initial chunk size of two seconds and 77.08% when it is six seconds.
- **Tor_normal is the most difficult to classify.** This is followed by SSH_noisy and SSH_normal is the easiest to classify. Previous researchers [22] have also observed that Tor is more difficult than SSH because all Tor cells have the same size.

The poor performance of our classifier on Tor_normal is due to large round-trip times: if we take a short initial chunk size, it may leave us with almost no data to classify. To mitigate this effect, we preprocess the training and testing data of Tor_normal, and remove the first ten packets in the network flow. Thus, each network flow will start from the 11th packet if there are fewer than ten packets in the first three seconds. We repeat the above experiment with

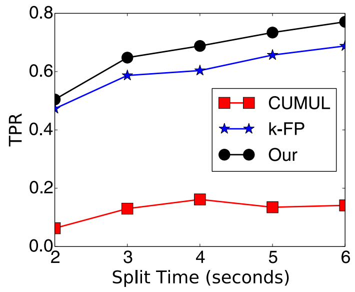

Figure 8: TPR of our classifier compared to *k*-FP and CUMUL while varying the initial chunk size between two seconds and six seconds on the Tor_normal dataset.

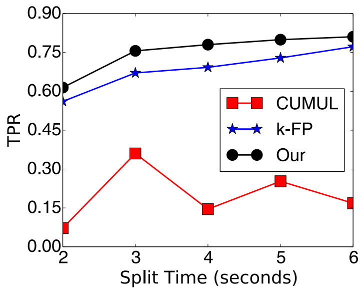

Figure 9: TPR of our classifier compared to *k*-FP and CUMUL while varying the initial chunk size between two seconds and six seconds on the modified Tor_normal dataset.

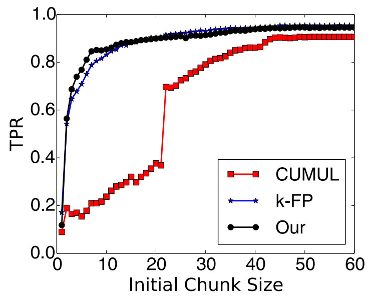

Figure 10: TPR of our classifier compared to *k*-FP and CUMUL while increasing the initial chunk up to the maximum 60 seconds on the Tor_normal dataset.

the modified Tor_normal dataset. Figure 9 shows that our modified dataset improves the accuracy of all three classifiers. When the split time is four seconds, we achieve 78% TPR, which is much better than Tor_normal. In particular, when the split time is six seconds, we achieve 81.04% TPR.

Furthermore, we measure how TPR changes when increasing the initial chunk size. We combine the training and testing subsets of Tor_normal, and then use 10-fold cross-validation to test the three classifiers. Note that, the reason why the setting here is different from the previous experiment setup in Section 6.4 is that we want to fairly compare the performance of identifying pages in the single tab and two-tab scenarios, where training on single tabs is required for the two-tab scenario, while 10-fold cross-validation used here

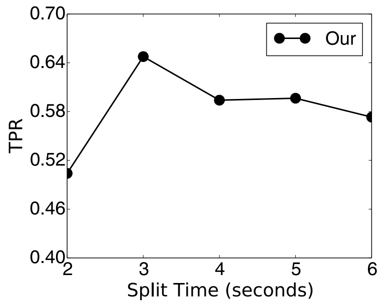

Figure 11: The TPR of different specified split time on Tor_two with two pages where true delay time is 3s

generates the distribution of instances close to the original datasets, which outputs reliable generalization error for the comparison of three attacks. As shown in Figure 10, our new WF classifier performs best when the initial chunk size is less than 17 seconds. *k*-FP is slightly better than our classifier if the whole network flow is used. When the initial chunk size is 10 seconds, our TPR is 85.4%, while we can have a TPR of 89.93% when the initial chunk size is 20 seconds. CUMUL performs well with the entire network flow and achieves 91% TPR. Our classifier is especially effective on a short initial chunk.

Evaluation with Open-World Setting. We also compare our classifier with *k*-FP and CUMUL on the more realistic open-world setting using 10-fold cross-validation. We vary the size of the unmonitored part from 500 instances to 2500 instances on SSH_normal training subset and Tor_normal training subset. Here, we use fingerprints of length 200 bytes recommended by [11] and set the neighbor of kNN to one that is the default setting in Hayes et al.'s code. For CUMUL, we use the same method as in closed-world setting.

As shown in Table 10, our attack has a TPR of 86.56% and an FPR of 0.52% when training on 2500 unmonitored web pages in SSH_normal. We show that the FPR of all three attacks decreases when we increase the number of unmonitored pages. Our attack achieves a higher TPR and lower FPR than both *k*-FP and CUMUL. When there are 500 unmonitored pages, we have a 90.23% TPR, which beats CUMUL by more than 20%, and *k*-FP by 3%.

We can achieve a TPR of 65.64% and FPR of 0.1% when training on 2500 unmonitored instances in Table 11. Although Tor_normal presents a greater challenge for classification than SSH_normal, our classifier nevertheless beats both CUMUL and *k*-FP when the initial chunk size is 3s. We can observe similar results with other initial chunk sizes. Furthermore, as shown in Figure 10, the TPR increases when the size of the initial chunk increases. The possible reason is that larger initial chunk sizes allows more data for classification and does not incur significant noise due to the slow connections on Tor.

The Impact of Wrong Split Time. In order to show the impact of the wrong split time (i.e., wrong initial chunks), we use the Tor_normal training subset with different assumed split points to train classifiers, where the delay ranges from two seconds to six seconds, and test such a classifier on the Tor dataset with two pages.

Table 10: TPR and FPR of our classifier compared to k-FP and CUMUL on SSH_normal with an initial chunk size of two seconds.

Page number	CUMUL		k-FP		Our	
	TPR	FPR	TPR	FPR	TPR	FPR
500	69.68%	28.8%	86.8%	10%	90.23%	6.2%
1500	66.28%	16.2%	86.36%	5.8%	88.84%	1.1%
2500	64.32%	11.48%	85.88%	4.48%	86.56%	0.52%

Table 11: TPR and FPR of our classifier compared to k-FP and CUMUL on Tor_normal with an initial chunk size of 3s.

Page number	CUMUL		k-FP		Our	
	TPR	FPR	TPR	FPR	TPR	FPR
500	58.8%	14.8%	62.56%	12%	66.04%	0.2%
1500	57.32%	6.8%	62.58%	10.86%	66.68%	0.1%
2500	56.36%	4.2%	62.56%	9.88%	65.64%	0.1%

We observe that, if we use data with a split time that is shorter than the real split time to train classifiers, the TPR of page classification decreases since we lose useful data for training. Figure 11 shows the TPR with different splits times on the Tor_two dataset. We observe that, when the true delay time is three seconds and we only use the first 2 seconds to train the classifier, the TPR decreases by around 5% since we lose useful information of the 3rd second. Similarly, if the split time is larger than the true delay time, we find that our TPR decreases since inappropriate split time incurs the features we extracted including the second page information. Fortunately, the accuracy is relatively stable when more noises are included in the training data. Our WF attack dynamically identifies the split time so that it does not waste the useful information or mix the noise into the features. Therefore, we can effectively construct the attack in practice.

6.5 Evaluation with More Than Two Tabs

We used Selenium to collect datasets with more than two tabs. We firstly load a random page and then request the subsequent pages with random delays. We obtain datasets with three tabs and four tabs. All datasets have 5000 samples. We use half of the datasets to train our page split classifier, and use another half to test. We observe that the split accuracy with more than two pages is relatively lower than that with two-tab pages. However, we can still obtain around 70% split accuracy. In particular, the split accuracy with various numbers of pages is similar. The possible reason is that, with the increase in the numbers of pages, the probability of overlapping the first page decreases. Note that, the existing attacks that almost fail to classify multi-web pages. Therefore, our attack can be still effective with more than two tabs pages.

7 CONCLUSION AND FUTURE WORK

In this paper, we described two new algorithms to relax the Single Page Assumption, an unrealistic assumption that all WF attacks relied on. For a client who visits two pages, where the amount of time between the two pages is demarcated by the split point, we consider an attacker who attempts to identify the first web page the client is visiting. Our strategy is first to develop a classifier that works with minimal amounts of data, and then to use such a classifier on an initial chunk of packets before the split point.

First, we showed that our new WF classifier could achieve a higher TPR compared to the previous best WF classifiers on an initial chunk of data. In the closed-world Tor scenario, we achieved a TPR of 77.08% on Tor when split time is six seconds and 93.88% on SSH only using the first two seconds of the initial chunk, beating CUMUL by Panchenko et al. and k-FP by Hayes et al. We found that our classifier became slightly less accurate when using more than two seconds of data on SSH, maybe due to the limited number of instances; however, we can achieve the highest TPR only using the first two seconds of initial chunk, this suggests that split finding is not necessary on SSH, and we should simply take two seconds of data to classify. It was still necessary to find the correct split point for the Tor scenario.

Second, we described a new split finding algorithm to identify the correct split point for the Tor scenario. Our algorithm uses BalanceCascade to resolve the class size imbalance between the false split class and the true split class. We use an ensemble of forests of regression trees to classify the data, where each forest of regression tree uses a novel gradient tree boosting technique by Chen and Guestrin called XGBoost. We found that our algorithm was able to outperform the previous state of the art, timekNN by Wang et al.

In a nutshell, when combining the split finding algorithm with our random forest classifier, we achieved an overall TPR of 64.94% on Tor; we achieve an overall TPR of 92.58%. Our work is, therefore, the first to show that it is still possible to perform WF against a client who visits multiple pages simultaneously.

ACKNOWLEDGMENTS

We would like to thank our shepherd Alexandros Kapravelos, and the anonymous reviewers for their insightful comments. This work was supported in part by the National Key R&D Program of China under Grant 2016YFB0800102, the National Natural Science Foundation of China under Grant 61572278, U1736209, 61602122, and 71731004, the Natural Science Foundation of Shanghai under Grant 16ZR1402200, and the Shanghai Pujiang Program under Grant 16PJ1400700. Qi Li is the corresponding author of this paper.

REFERENCES

[1] Pablo Bermejo, José A Gámez, and José M Puerta. 2014. Speeding up Incremental Wrapper Feature Subset Selection with Naive Bayes Classifier. *Knowledge-Based Systems* 55 (2014), 140–147.

[2] George Dean Bissias, Marc Liberatore, David Jensen, and Brian Neil Levine. 2005. Privacy vulnerabilities in encrypted HTTP streams. In *International Workshop on Privacy Enhancing Technologies*. Springer, 1–11.

[3] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (01 Oct 2001), 5–32. https://doi.org/10.1023/A:1010933404324

[4] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. 1984. *Classification and regression trees.* CRC press.

[5] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. 2014. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 227–238.

[6] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. 2012. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *Proceedings of the 2012 ACM conference on Computer and communications security.* ACM, 605–616.

[7] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining.* ACM, 785–794.

[8] Thomas Cover and Peter Hart. 1967. Nearest neighbor pattern classification. *IEEE transactions on information theory* 13, 1 (1967), 21–27.

[9] Kevin P Dyer, Scott E Coull, Thomas Ristenpart, and Thomas Shrimpton. 2012. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *Security and Privacy (SP), 2012 IEEE Symposium on.* IEEE, 332–346.

[10] Xiaodan Gu, Ming Yang, and Junzhou Luo. 2015. A novel Website Fingerprinting attack against multi-tab browsing behavior. In *Computer Supported Cooperative Work in Design (CSCWD), 2015 IEEE 19th International Conference on.* IEEE, 234–239.

[11] Jamie Hayes and George Danezis. 2016. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique.. In *USENIX Security Symposium.* 1187–1203.

[12] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. 2009. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-bayes Classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security.* ACM, 31–42.

[13] Andrew Hintz. 2002. Fingerprinting Websites Using Traffic Analysis. In *International Workshop on Privacy Enhancing Technologies.* Springer, 171–178.

[14] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. 2014. A Critical Evaluation of Website Fingerprinting Attacks. In *Proceedings of the 2014 ACM Conference on Computer and Communications Security.* ACM, 263–274.

[15] Andy Liaw and Matthew Wiener. 2002. Classification and Regression by randomForest. *R news* 2, 3 (2002), 18–22.

[16] Marc Liberatore and Brian Neil Levine. 2006. Inferring the Source of Encrypted HTTP Connections. In *Proceedings of the 13th ACM conference on Computer and communications security.* ACM, 255–263.

[17] Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou. 2009. Exploratory Undersampling for Class-imbalance Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39, 2 (2009), 539–550.

[18] Liming Lu, Ee-Chien Chang, and Mun Chan. 2010. Website Fingerprinting and Identification Using Ordered Feature Sequences. *Computer Security–ESORICS 2010* (2010), 199–214.

[19] Xiapu Luo, Peng Zhou, Edmond WW Chan, Wenke Lee, Rocky KC Chang, and Roberto Perdisci. 2011. HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows.. In *NDSS.*

[20] Kevin P Murphy. 2006. Naive Bayes Classifiers. *University of British Columbia* 18 (2006).

[21] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. 2016. Website Fingerprinting at Internet Scale.. In *NDSS.*

[22] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. 2011. Website Fingerprinting in Onion Routing based Anonymization Networks. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society.* ACM, 103–114.

[23] F Donelson Smith, Félix Hernández Campos, Kevin Jeffay, and David Ott. 2001. What TCP/IP protocol headers can tell us about the web. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 29. ACM, 245–256.

[24] Qixiang Sun, Daniel R Simon, Yi-Min Wang, Wilf Russell, Venkata N Padmanabhan, and Lili Qiu. 2002. Statistical Identification of Encrypted Web Browsing Traffic. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on.* IEEE, 19–30.

[25] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective Attacks and Provable Defenses for Website Fingerprinting.. In *USENIX Security Symposium.* 143–157.

[26] Tao Wang and Ian Goldberg. 2013. Improved Website Fingerprinting on Tor. In *Proceedings of the 12th annual ACM workshop on Privacy in the electronic society.* ACM, 201–212.

[27] Tao Wang and Ian Goldberg. 2016. On Realistically Attacking Tor with Website Fingerprinting. *Proceedings on Privacy Enhancing Technologies* 2016, 4 (2016), 21–36.

[28] Charles V Wright, Scott E Coull, and Fabian Monrose. 2009. Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis.. In *NDSS.*

A THE REST FEATURES IN FEATURE SET

This section shows the other features used in our attack as well. Together with the features shown in Section 5, our attack can achieve better performance than CUMUL and *k*-FP.

- The cumulative size of packets without MTU size (CSOPWMS). The feature is similar with CRFONF, deleting the packet bigger than 1448 from the network flow, and the number of samples is five.
- The quantity of incoming in the first 20 packets of network flow.
- URL_length. URL_length is the size of the first outgoing packet which is a request to the server's HTML document.
- Statistics of the quantity of packets. The quantity of total packets and incoming packets. The quantity of incoming packets as fraction of total packets.
- Statistics of size of packets. The total size of outgoing packets, the total incoming size of packets as fraction of total packets.
- the quantity of incoming & outgoing packets and the size of outgoing packets rounded to the nearest multiple of 100.
- Document length. If the second outgoing packet is sent at time t, we take all the incoming packets before $t + RTT$ as the document length. The HTML document contains text and objects links which will be loaded by browser, whose size is a more constant value compared to changeable object such as img. This may not be applied in Tor, because Tor may send multiple outgoing packets in a row at the start.
- the quantity and the transmission speed of incoming and outgoing packets. For instance, to compute the speed of total number, for each recorded packet, we extract a list using 1 to divide the inter-arrival time, and sampling the list to 20 samples.
- Vector inner product using packet length. Similar with FLLD, we compare the distance of two instances with inner vector product using the bag of packets length.

B FEATURE SELECTION

The following tables illustrate the features selected from various datasets. As shown in Table 12, most FLLD features are included in the feature subset of SSH_normal. However, the features related to the transmission speed of packets are much less included. In addition, the time features have more importance, which almost takes 10% of the total features. The Tor_normal dataset only uses one CSOP feature and has the second smallest subset among all the datasets when the split time is two seconds. While the features related to outgoing packets play a key role in Tor_normal when the split time is three seconds. There are around 20% out of the total features. The HTTPOS split and Traffic morphing datasets use similar features to that used in SSH_normal. The Decoy pages dataset has the largest subset among all the datasets. The BuFLO and Decoy pages datasets use many features related to time and FLLD.

Table 12: The most useful features selected from the SSH_normal dataset.

No.	Description of Features
1	RTT
2	average_packet_number_before_every_incom-fing_packet
3	the 2-4th burst_number_packet
4	the 1-5th burst_size_packet
5	the 2-6th, 8-11th, 29th, 98-99th cumulative_packet
6	the 5th cumulative_without_mtu_packet
7	first_quartile_of_outgoing_transmission_time
8	first_quratile_of_incoming_transmission_time
9	first_quratile_of_transmission_time
10	the 1st in_size_speed_packet
11	incoming_packet_number_ratio_in_the_first_20_packets
12	incoming_size
13	maximun_inter_arrival_time_of_incoming_packets
14	maximun_inter_arrival_time_of_total_packets
15	minimum_inter_arrival_time_of_incoming_packets
16	the 1st number_speed_packet
17	the 3-5th out_number_speed_packet
18	the 4th out_size_speed_packet
19	outgoing_packet_number
20	outgoing_packet_number_in_the_first_20_packets
21	outgoing_packet_number_ratio_in_the_first_20_packets
22	outgoing_packet_size_ratio
23	rounded_document_length
24	rounded_incoming_size
25	second_quartile_of_transmission_time
26	total_size
27	the 1-4th, 6th,8th, 12th, 16-17th, 19-23th, 26th, 28-31st, 33-35th, 39-40th, 42th, 44th, 46-50th website_similarity_by_fast_edit_distance 16th, 18th, 31st, 35th website_similarity_by_jaccard

Table 13: The most useful features selected from the Tor_normal dataset when the split time is two seconds.

No.	Description of Features
1	average_packet_number_before_every_incoming_packet
2	standard_deviation_of_packet_number_before_every_incoming_packet
3	minimum_inter_arrival_time_of_outgoing_packets
4	second_quartile_of_transmission_time
5	rounded_incoming_size
6	the 2th out_number_speed_packet
7	the 3th, 6th, 10th in_size_speed_packet
8	the 2-3th out_size_speed_packet
9	the 2th cumulative_packet
10	the 1st, 3th burst_size_packet
11	the 1-3th burst_number_packet
12	the 21st, 37th, 40th website_similarity_by_vector
13	the 7th, 10th, 14th, 22th, 24th, 31st, 33-34th, 38th, 44th, 46th website_similarity_by_fast_edit_distance

Table 14: The most useful features selected from the Tor_normal dataset when the split time is three seconds

No.	Description of Features
1	outgoing_size
2	outgoing_packet_number
3	outgoing_packet_number_in_the_first_20_packets
4	outgoing_packet_number_ratio_in_the_first_20_packets
5	outgoing_packet_number_ratio_in_the_last_20_packets
6	standard_deviation_of_packet_number_before_every_incoming_packet
7	standard_deviation _of_packet_number_before_every_outgoing_packet
8	average_inter_arrival_time_of_outgoing_packets
9	std_inter_arrival_time_of_outgoing_packets
10	rounded_document_length
11	rounded_outgoing_size
12	the 3th out_number_speed_packet
13	the 1st size_speed_packet
14	the 1st, 3-4th, 13th in_size_speed_packet
15	the 2th, 3th, 5th out_size_speed_packet
16	the 2-5th, 7th, 15th, 51st cumulative_packet
17	the 1st,3-5th burst_size_packet
18	the 1-5th burst_number_packet
19	the 15th, 17th, 19-20th, 25th, 30th, 33-35th, 43th, 45-46th, 50th website_similarity_by_fast_edit_distance

Table 15: The most useful features select from the HTTPOS split dataset when the split time is two seconds.

No.	Description of Features
1	RTT
2	the 3-4th, 19th burst_number_packet
3	the 2-6th, 14-15th, 18th burst_size_packet
4	the 2-7th, 9th, 12th, 14-15th, 17th, 85th, 100th cumulative_packet
5	the 1st, 3th cumulative_without_mtu_packet
6	first_quratile_of_incoming_transmission_time
7	first_quratile_of_transmission_time
8	the 1st in_number_speed_packet
9	the 1st in_size_speed_packet
10	incoming_packet_number_ratio_in_the_fir-st_20_packets
11	incoming_packet_size_ratio
12	incoming_size
13	maximun_inter_arrival_time_of_incoming_pa-ckets
14	minimum_inter_arrival_time_of_incoming_pa-ckets
15	number_speed_packet
16	out_number_speed_packet
17	the 4th out_size_speed_packet
18	the 7th out_size_speed_packet
19	outgoing_packet_number_ratio
20	outgoing_packet_number_ratio_in_the_fir-st_20_packets
21	outgoing_packet_size_ratio
22	rounded_document_length
23	rounded_incoming_size
24	second_quartile_of_incoming_transmissio-n_time
25	second_quartile_of_outgoing_transmissi-on_time
26	third_quartile_of_ougoing_transmission_time
27	total_incoming_transmission_time
28	the 5-8th, 16-21st, 23th, 27th, 29-30th, 32th, 34-38th, 42th, 44th, 46th, 48th web-site_similarity_by_fast_edit_distance
29	the 14-15th, 28th, 39th, 45th web-site_similarity_by_jaccard
30	the 2th, 9th, 24th website_similarity_by_vector

Table 16: The most useful features selected from the Traffic morphing dataset.

No.	Description of Features
1	RTT
2	average_packet_number_before_every_ou-tgoing_packet
3	the 2th, 5th, 7-8th, 13th, 16th, 18th burst_number_packet
4	the 2-6th, 9th, 11th, 20th burst_size_packet
5	the 2-3th, 6-7th, 12th, 14th, 15th, 24-25th, 36-37th, 54-55th, 64th, 79th, 96-97th cumulative_packet
7	the 2th cumulative_without_mtu_packet
8	first_quratile_of_incoming_transmission_time
9	first_quratile_of_transmission_time
10	the 1st in_number_speed_packet
11	the 1st, 5th, 7th, 12-13th, 15th, 17th in_size_speed_packet
12	incoming_packet_number_in_the_fir-st_20_packets
13	incoming_packet_number_ratio_in_the_fir-st_20_packets
14	incoming_packet_number_ratio_in_the_la-st_20_packets
15	incoming_packet_size_ratio
16	maximun_inter_arrival_time_of_incoming_p-ackets
17	maximun_inter_arrival_time_of_outgoing_p-ackets
18	maximun_inter_arrival_time_of_total_packets
19	minimum_inter_arrival_time_of_incoming_pa-ckets
20	the 4th, 6th, 24th out_number_speed_packet
21	the 4th, 6th, 25th out_size_speed_packet
22	outgoing_packet_number_in_the_first_20_p-ackets
23	outgoing_packet_number_in_the_last_20_pa-ckets
24	outgoing_packet_number_ratio
25	outgoing_packet_number_ratio_in_the_fir-st_20_packets
26	outgoing_packet_number_ratio_in_the_la-st_20_packets
27	rounded_document_length
28	rounded_incoming_size
29	rounded_outgoing_size
30	second_quartile_of_incoming_transmission_ti-me
31	the 2-3th size_speed_packet
32	standard_deviation_of_packet_num-ber_before_every_outgoing_packet
33	third_quartile_of_incoming_transmission_time
34	third_quartile_of_transmission_time
35	total_incoming_transmission_time
36	the 1st, 6th, 14th, 22th, 27th, 33th, 36th web-site_similarity_by_fast_edit_distance

Table 17: The most useful features selected from the Decoy pages dataset.

No.	Description of Features
1	RTT
2	URL_length
3	the 2-7th, 11th, 19th burst_number_packet
4	the 1-4th, 19th burst_size_packet
5	the 2-3th, 42th, 52th, 55-56th cumulative_packet
6	the 1st, 5th cumulative_without_mtu_packet
7	first_quartile_of_outgoing_transmission_time
8	first_quratile_of_incoming_transmission_time
9	first_quratile_of_transmission_time
10	the 1st in_number_speed_packet
11	the 1st in_size_speed_packet
12	incoming_packet_number
13	incoming_packet_number_in_the_first_20_packets
14	incoming_packet_number_ratio
15	the 3th out_number_speed_packet
16	outgoing_packet_number_in_the_first_20_packets
17	outgoing_packet_size_ratio
18	outgoing_packet_number_ratio_in_the_first_20_packets
19	outgoing_packet_size_ratio
20	rounded_document_length
21	rounded_incoming_size
22	second_quartile_of_incoming_transmission_time
23	second_quartile_of_outgoing_transmission_time
24	second_quartile_of_transmission_time
25	standard_deviation_of_packet_number_before_every_outgoing_packet
26	third_quartile_of_incoming_transmission_time
27	third_quartile_of_ougoing_transmission_time
28	third_quartile_of_transmission_time
29	the 1st, 3-7th, 9-10th, 12th, 14-18th, 21-25th, 28th, 30-35th, 36-37th, 39-40th, 42th, 44th, 46-50th website_similarity_by_fast_edit_distance
30	the 3th, 17-18th, 23th, 49th website_similarity_by_jaccard
31	the 5th, 26th, 38th, 42th, 46th website_similarity_by_vector

Table 18: The most useful features selected from the BuFLO dataset.

No.	Description of Features
1	first_quratile_of_transmission_time
2	the 1st out_size_speed_packet
3	the 1st in_size_speed_packet
4	third_quartile_of_ougoing_transmission_time
5	third_quartile_of_incoming_transmission_time
6	99th cumulative_packet
7	the 7th, 9th, 19th, 28th, 3, 39-41th, 49th website_similarity_by_fast_edit_distance