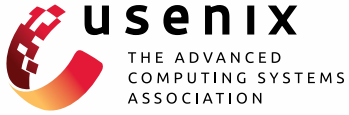

Effective Attacks and Provable Defenses for Website Fingerprinting

Tao Wang, ***University of Waterloo;*** **Xiang Cai, Rishab Nithyanand, and Rob Johnson,** ***Stony Brook University;*** **Ian Goldberg,** ***University of Waterloo***

https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/wang_tao

This paper is included in the Proceedings of the 23rd USENIX Security Symposium.

August 20–22, 2014 • San Diego, CA

ISBN 978-1-931971-15-7

Open access to the Proceedings of the 23rd USENIX Security Symposium is sponsored by USENIX

Effective Attacks and Provable Defenses for Website Fingerprinting

Tao Wang[1] Xiang Cai[2] Rishab Nithyanand[2] Rob Johnson[2] Ian Goldberg[1]

[1]University of Waterloo
{t55wang,iang}@cs.uwaterloo.ca

[2]Stony Brook University
{xcai,rnithyanand,rob}@cs.stonybrook.edu

Abstract

Website fingerprinting attacks allow a local, passive eavesdropper to identify a user's web activity by leveraging packet sequence information. These attacks break the privacy expected by users of privacy technologies, including low-latency anonymity networks such as Tor. In this paper, we show a new attack that achieves significantly higher accuracy than previous attacks in the same field, further highlighting website fingerprinting as a genuine threat to web privacy. We test our attack under a large open-world experimental setting, where the client can visit pages that the attacker is not aware of. We found that our new attack is much more accurate than previous attempts, especially for an attacker monitoring a set of sites with low base incidence rate. We can correctly determine which of 100 monitored web pages a client is visiting (out of a significantly larger universe) at an 85% true positive rate with a false positive rate of 0.6%, compared to the best of 83% true positive rate with a false positive rate of 6% in previous work.

To defend against such attacks, we need provably effective defenses. We show how simulatable, deterministic defenses can be provably private, and we show that bandwidth overhead optimality can be achieved for these defenses by using a supersequence over anonymity sets of packet sequences. We design a new defense by approximating this optimal strategy and demonstrate that this new defense is able to defeat any attack at a lower cost on bandwidth than the previous best.

1 Introduction

Privacy technologies are becoming more popular: Tor, a low-latency anonymity network, currently has 500,000 daily users and the number has been growing [21]. However, users of Tor are vulnerable to *website fingerprinting* attacks [4, 17, 23]. Users of other privacy technologies such as SSH tunneling, VPNs and IPsec are also vulnerable to website fingerprinting [10].

When a client browses the web, she reveals her destination and packet content to intermediate routers, which are controlled by ISPs who may be susceptible to malicious attackers, eavesdroppers, and legal pressure. To protect her web-browsing privacy, the client would need to encrypt her communication traffic and obscure her destinations with a proxy such as Tor. Website fingerprinting refers to the set of techniques that seek to re-identify these clients' destination web pages by passively observing their communication traffic. The traffic will contain packet lengths, order, and timing information that could uniquely identify the page, and website fingerprinting attacks use machine classification to extract and use this information (see Section 2).

A number of attacks have been proposed that would compromise a client's expected privacy, and defenses have been proposed to counter these attacks (see Section 3). Most previous defenses have been shown to fail against more advanced attacks [4, 6, 15]; this is because they were evaluated only against specific attacks, with no notion of provable effectiveness (against all possible attacks). In this paper, we will show an attack that further highlights the fact that clients need a provably effective defense, for which an upper bound on the accuracy of any possible attack can be given. We will then show how such a defense can be constructed. Only with a provably effective defense can we be certain that clients are protected against website fingerprinting.

The contributions of our paper are as follows:

1. We propose a significantly improved attack that achieves a higher accuracy with a training and testing time that is orders of magnitude lower than the previous best. Our attack is a k-Nearest Neighbour classifier applied on a large feature set with weight adjustment. Our attack is designed to find flaws in defenses and achieve high success rates even with those defenses, and we demonstrate that

several known defenses have almost no impact on our attack. We describe this attack in Section 4.

2. Using this attack, we tackle a large open-world problem, in which the attacker must determine which of 100 monitored pages the client is visiting, but the client can visit a large number of pages that the attacker cannot train on. We demonstrate that the attack is still truly effective in this realistic scenario in Section 5, and that it outperforms the previous best attack by Wang and Goldberg [23] (which we call OSAD) on the same data set.

3. We show that simulatable, deterministic defenses can be turned into provably private defenses in our model. In our model, we consider a defense to be successful only if it produces packet sequences that are identical (in time, order, and packet lengths) to packet sequences from different web pages. This strong notion of indistinguishability of packet sequences yields our provably private defenses. We found that the bandwidth-optimal simulatable, deterministic defense is to transmit packets using supersequences over anonymity sets. We construct a principled defense using an approximation of the smallest common supersequence problem and clustering techniques in Section 6 and evaluate it in Section 7.

We follow up with a discussion on realistic applicability and reproducibility of our results in Section 8 and conclude in Section 9.

2 Basics

2.1 Website Fingerprinting on Tor

Website fingerprinting (WF) refers to the process of attempting to identify a web-browsing client's behaviour—specifically, which web pages she is visiting—by observing her traffic traces. We assume that the client is using a proxy to hide her true destination, as well as encryption to hide her packet contents, as without these basic defenses she reveals her destination to a trivial eavesdropper. Users of Tor have these defenses.

More recent attacks can successfully perform website fingerprinting with an attacker that only has local observation capacity; i.e. the attacker merely observes the traffic traces of the client without any interference. The attacker is located on the client's network, such as the client's ISP, or he has gained control of some router near the client. Attacks requiring more capabilities have been proposed, such as attacks which leverage active traffic-shaping strategies [8], remote ping detection [9] and, sometimes, involve tampering with the client's device [12]. Our attack achieves high accuracy with only a local, passive attacker.

In general, the attacker's strategy is as follows. The attacker collects packet traces from several web pages that he is interested in monitoring. Then, the attacker observes packet traces generated by the client during her web browsing, and compares these traces with the ones he collected by performing supervised classification. We note two assumptions that all previous works on WF have made of the attacker:

1. Well-defined packet traces. It is assumed that the attacker knows where the packet trace of a single page load starts and ends. If the client takes much longer to load the next page after the current one is loaded, this assumption can be justified.

2. No other activity. We assume the client is not performing any other activity that could be confused for page-loading behaviour, such as downloading a file.

These assumptions are used by all previous works on WF as they simplify the problem, though it should be noted that these assumptions are advantageous for the attacker. We discuss how the attacker can carry out a successful attack without these assumptions in Section 8.

Website fingerprinting is harder on Tor than simple SSH or VPN tunneling [10]. This is because Tor uses cell padding, such that data is sent in fixed-size (512-byte) cells. In addition, Tor has background noise (circuit construction, SENDME packets, etc.) which interferes with website fingerprinting [23]. As Tor has a large user base and an extensive architecture upon which defenses can be applied, recent works and our work are interested in attacking and defending Tor, especially as Tor developers remain unconvinced that website fingerprinting poses a real threat [19].

2.2 Classification

Given a packet sequence, the attacker learns the client's destination web page with a classification algorithm (classifier). The attacker first gathers packet sequences of known pages that he is interested in monitoring (the training set). This is known as supervised training as the true labels of these packet sequences are known to the attacker. We can test the effectiveness of such a classifier by applying it to a data set of packet sequences that the attacker did not train on (the testing set), and measuring the accuracy of the classifier's predictions.

Central to the classifier is a notion of distance between packet sequences. A larger distance indicates that the two packet sequences are less likely to be from the same

page. Previous authors have used varying formulae for distance, ranging from comparing the occurrence counts of unique packet lengths to variations of Levenshtein distance. The distance used reflects how features are used to distinguish web pages. These features are, explicitly or implicitly, extracted from packet sequences to compare them with each other.

Our attack is based on the important observation that a class representing a web page is multi-modal. Several factors cause a web page to vary: network conditions, random advertisements and content, updating data over time, and unpredictable order of resources. Client configuration may also affect page loading.[1] An attacker can deal with multi-modal data sets by gathering enough data to have representative elements from each mode. For example, an attacker can gather two modes of a page, one for low-bandwidth connections, and another for high-bandwidth connections.[2] We use a classifier designed for multi-modal classes, for which different modes of the class do not need to have any relationship with each other.

3 Related Work

This section surveys the related work on website fingerprinting (WF). We classify attacks into those which depend on revealed resource lengths (HTTP 1.0), revealed packet lengths (HTTP 1.1, VPNs, SSH tunneling, etc.), and hidden packet lengths (Tor). We also survey the previous work on defenses in this section.

3.1 Resource length attacks

In HTTP 1.0, web page resources (images, scripts, etc.) are each requested with a separate TCP connection. This implies that an attacker who is able to distinguish between different connections can identify the total length of each resource. The earliest attacks were performed in this scenario: Cheng et al. in 1998 [5], Sun et al. in 2002 [20], and Hintz in 2003 [11]. These works showed that observing resource lengths can help identify a page. HTTP 1.1 uses persistent connections, and therefore more recent browsers and privacy technologies are not susceptible to resource length attacks.

3.2 Unique packet length attacks

Liberatore and Levine in 2006 [14] showed how unique packet lengths are a powerful WF feature with two attacks: one using the Jaccard coefficient and another using the Naïve Bayes classifier. Under the first attack, the classifier mapped each packet sequence to its set of unique packet lengths (discarding ordering and frequency). Then, it used the Jaccard coefficient as a measurement of the distance between two packet sequences. The Naïve Bayes classifier used packet lengths and their occurrence frequencies as well, but also discarded ordering and timing. The Naïve Bayes assumption is that the occurrence probabilities of different packet lengths are independent of each other. Later, Herrmann et al. [10] proposed a number of improvements to this attack by incorporating techniques from text mining.

Bissias et al. in 2006 [2] published an attack based on cross-correlation with interpacket timings, but it is less accurate than the Naïve Bayes attacks. Lu et al. in 2010 [15] published an attack that heavily focuses on capturing packet burst patterns with packet ordering, discarding packet frequencies and packet timing.

3.3 Hidden packet length attacks

Herrmann et al. were not able to successfully perform WF on Tor [17], where unique packet lengths are hidden by fixed-size Tor cells. In 2009, Panchenko et al. [17] showed an attack that succeeded against web-browsing clients that use Tor. As unique packet lengths are hidden on Tor, Panchenko et al. used other features, which are processed by a Support Vector Machine (SVM). These features attempted to capture burst patterns, main document size, ratios of incoming and outgoing packets, and total packet counts, which helped identify a page. Dyer et al. in 2012 [6] used a similar but smaller set of features for a variable n-gram classifier, but their classifier did not perform better in any of the scenarios they considered.

Cai et al. in 2011 improved the accuracy of WF on Tor. Using the edit distance to compare packet sequences, they modified the kernel of the SVM and showed an attack with significantly increased accuracy on Tor [4]. Wang and Goldberg in 2013 further improved the accuracy of Cai et al.'s scheme on Tor by modifying the edit distance algorithm [23], creating OSAD. These modifications were based on observations on how web pages are loaded. As it is the current state of the art under the same attack scenario, we will compare our attack to OSAD.

3.4 Defenses

Defenses are applied on the client's connection in order to protect her against website fingerprinting attacks. We present a new classification of defenses in this section. First, defenses can be "simulatable" or "non-simulatable". A simulatable defense can be written as a defense function D that takes in a packet sequence and

[1] On the Tor Browser changing the browser configuration is discouraged as it makes browser fingerprinting easy.

[2] Data collection on Tor will naturally result in such a situation because of random circuit selection.

outputs another packet sequence. The function does not look at the true contents of the packets, but only their length, direction and time. An advantage of simulatable defenses is the implementation cost, as non-simulatable defenses would need to be implemented on the browser and would have access to client data, which may be difficult for some clients to accept. The implementation of a simulatable defense requires no more access to information than a website fingerprinting attacker would typically have.

Secondly, defenses can be "deterministic" or "random"—for deterministic defenses the function D always returns the same packet sequence for each input packet sequence p.[3] Our goal is to design a provably private defense that has an upper bound on the accuracy of any attack. Random defenses (noise) have the disadvantage that choosing a good covering is not guaranteed. An attacker that can link together different page loads can partially remove the noise. Furthermore, implementations of random defenses must be careful so that noise cannot be easily distinguished from real packets.

Non-simulatable, random: This includes Tor's request order randomization defense. Responding to Panchenko's attack, Tor developers decided to enable pipelining on Tor and randomize pipeline size and request orders [18]. The randomization was further increased after OSAD [19]. We test our attack against the more randomized version that is built into Tor Browser Bundle 3.5.

Non-simulatable, deterministic: This includes portions of HTTPOS [16]. The HTTPOS defense is built into the client's browser, allowing the client to hide unique packet lengths by sending an HTTP range request strategically.

Simulatable, random: This includes traffic morphing [24], which allows a client to load a web page using a packet size distribution from a *different* page, and Panchenko's background noise [17], where a decoy page is loaded simultaneously with the real page to hide the real packet sequence.

Simulatable, deterministic: This includes packet padding, which is done on Tor, and BuFLO, presented and analyzed by Dyer et al. [6]. BuFLO sends data at a constant rate in both directions until data transfer ends. In this work, we will show that defenses in this category can be made to be provably private,[4] and we will show such a defense with a much lower overhead than BuFLO.

[3] Using a random procedure to learn D does not make D itself random.

[4] BuFLO is not provably private on its own.

4 Attack

In this section, we describe our new attack, which is designed to break website fingerprinting defenses. Our attack is based on the well-known k-Nearest Neighbours (k-NN) classifier, which we briefly overview in Section 4.1. The attack finds flaws in defenses by relying on a large feature set, which we describe in Section 4.2. We then train the attack to focus on features which the defense fails to cover and which therefore remain useful for classification. We describe the weight adjustment process in Section 4.3.

4.1 k-NN classifier

k-NN is a simple supervised machine learning algorithm. Suppose the training set is S_{train} and the testing set is S_{test}. The classifier is given a set of training points (packet sequences) $S_{train} = \{P_1, P_2, \ldots\}$. The training points are labeled with classes (the page the packet sequence was loaded from); let the class of P_i be denoted $C(P_i)$. Given a testing point $P_{test} \in S_{test}$, the classifier guesses $C(P_{test})$ by computing the distance $D(P_{test}, P_{train})$ for each $P_{train} \in S_{train}$. The algorithm then classifies P_{test} based on the classes of the k closest training points.

Despite its simplicity, the k-NN classifier has a number of advantages over other classifiers. Training involves learning a distance between pairs of points; the classifier could use a known (e.g. Euclidean) distance, though selecting the distance function carefully can greatly improve the classification accuracy. Testing time is very short, with a single distance computation to each training point. Multi-modal sets can be classified accurately, as the classifier would only need to refer to a single mode of each training set.

The k-NN classifier needs a distance function d for pairs of packet sequences. The distance is non-trivial for packet sequences. We want the distance to be accurate on simple encrypted data without extra padding, but also accurate when defenses are applied that remove features from our available feature set. We therefore start with a large feature set $F = \{f_1, f_2, \ldots\}$. Each feature is a function f which takes in a packet sequence P and computes $f(P)$, a non-negative number. Conceptually, each feature is designed such that members of the same class are more likely to have similar features than members of different classes. We give our feature set in Section 4.2. The distance between P and P' is computed as:

$$d(P,P') = \sum_{1 \le i \le |F|} w_i |f_i(P) - f_i(P')|$$

The weights $W = \{w_1, w_2, \ldots, w_{|F|}\}$ are learned as in Section 4.3, where we describe how the weights for uninformative features (such as one that a defense success-

fully covers) are reduced. As weight learning proceeds, the k-NN distance comes to focus on features that are useful for classification.

We tried a number of other distances, including the edit distance used by Cai et al. [4] and OSAD, which they used to compute the kernel of SVMs. The results for using their distances on the k-NN classifier are similar to those using the SVM. As we shall see in Section 5, using our proposed distance allows a significant improvement in accuracy over these distances, with or without extra defenses.

4.2 Feature set

Our feature set is intended to be diverse. The construction of the feature set is based on prior knowledge of how website fingerprinting attacks work and how defenses fail.

Our feature set includes the following:

- General features. This includes total transmission size, total transmission time, and numbers of incoming and outgoing packets.

- Unique packet lengths. For each packet length between 1 and 1500, and each direction, we include a feature which is defined as 1 if it occurs in the data set and 0 if it does not. This is similar to the algorithms used by Liberatore and Levine [14] and Herrmann et al. [10], where the presence of unique packet lengths is an important feature. These features are not useful when packet padding is applied, as on Tor.

- Packet ordering. For each outgoing packet, we add, in order, a feature that indicates the total number of packets before it in the sequence. We also add a feature that indicates the total number of incoming packets between this outgoing packet and the previous one. This captures the burst patterns that helped Cai et al. achieve their high accuracy rates.

- Concentration of outgoing packets. We count the number of outgoing packets in non-overlapping spans of 30 packets, and add that as a feature. This indicates where the outgoing packets are concentrated without the fineness (and volatility) of the packet ordering features above.

- Bursts. We define a burst of outgoing packets as a sequence of outgoing packets, in which there are no two adjacent incoming packets. We find the maximum and mean burst length, as well as the number of bursts, and add them as features.

- Initial packets. We also add the lengths of the first 20 packets (with direction) in the sequence as features.

Some feature sets, such as packet ordering, have variable numbers of features. We define a maximum number of features for the set, and if the packet sequence does not have this many features, we pad with a special character (X) until it reaches the maximum number. Recall that our distance is the weighted sum of absolute differences between features; let us denote the difference as $d_{f_i}(P,P')$. For each feature f_i, if at least one of the two values is X, then we define $d_{f_i}(P,P')$ to be 0, such that the difference is ignored and does not contribute to the total distance. Otherwise, we compute the difference as usual.

We treat all features equally. However, we note that as the general features are amongst the strongest indicators of whether or not two packet sequences belong to the same mode of a page, we could use them with a search algorithm to significantly reduce training and testing time (i.e. reject pages with significantly different values in the general feature without computing the whole distance).

The total number of features is close to 4,000 (3,000 of which are just for the unique packet lengths). If a defense covers some features and leaves others open (e.g. traffic morphing retains total transmission size and burst features), our algorithm should be successful in adjusting weights to focus on useful features.

We design our attack by drawing from previous successful attacks, while allowing automatic defense-breaking. In particular, we note that there exists a choice of weights for which our attack uses a similar distance metric as the attacks proposed by Cai et al. [4] and Wang and Goldberg [23], as well as the Jaccard coefficient by Liberatore and Levine [14]. However, we will find better choices of weights in the next subsection. We drew the inspiration for some features from the work by Panchenko et al. [17], in particular, those concerning the start of the page (which may indicate the size of the HTML document). We note that unlike Panchenko et al. [17], we do not add the entire packet sequence as features.

4.3 Weight initialization and adjustment

In this subsection, we describe how we learn $w_1, w_2, \ldots, w_{|F|}$, the weights that determine our distance computation. The values of these weights determine the quality of our classifier. We learn the weights using an iterative, local weight-learning process as follows. The weight-learning process is carried out for R rounds (we will see how the choice of R affects the accuracy later). For each round, we focus on a point $P_{train} \in S_{train}$ (in or-

der), performing two steps: the weight recommendation step and the weight adjustment step.

Weight recommendation. The objective of the weight recommendation step is to find the weights that we want to reduce. During the weight recommendation step, the distances between P_{train} and all other $P' \in S_{train}$ are computed. We then take the closest k_{reco} points (for a parameter k_{reco}) within the same class $S_{good} = \{P_1, P_2, \ldots\}$ and the closest k_{reco} points within all other classes $S_{bad} = \{P'_1, P'_2, \ldots\}$; we will focus only on those points.

We denote $d(P,S)$, where S is a set of packet sequences, as the sum of the distances between P and each sequence in S.

Let us denote

$$d_{maxgood_i} = \max(\{d_{f_i}(P_{train}, P) | P \in S_{good}\})$$

For each feature, we compute the number of relevant bad distances, n_{bad_i}, as

$$n_{bad_i} = |\{P' \in S_{bad} | d_{f_i}(P_{train}, P') \leq d_{maxgood_i}\}|$$

This indicates how bad feature f_i is in helping to distinguish S_{bad} from S_{good}. A large value of n_{bad_i} means that feature f_i is not useful at distinguishing members of P_{train}'s class from members of other classes, and so the weight of f_i should be decreased; for example, features perfectly covered by a defence (such as unique packet lengths in Tor) will always have $n_{bad_i} = k_{reco}$, its maximum possible value. Conversely, small values of n_{bad_i} indicate helpful features whose weights should be increased.

Weight adjustment. We adjust the weights to keep $d(P_{train}, S_{bad})$ the same while reducing $d(P_{train}, S_{good})$. Then, for each i such that $n_{bad_i} \neq \min(\{n_{bad_1}, n_{bad_2}, \ldots, n_{bad_{|F|}}\})$, we reduce the weight by $\Delta w_i = w_i \cdot 0.01$. We then increase all weights w_i with $n_{bad_i} = \min(\{n_{bad_1}, n_{bad_2}, \ldots, n_{bad_{|F|}}\})$ equally such that $d(P_{train}, S_{bad})$ remains the same.

We achieved our best results with two more changes to the way weights are reduced, as follows:

- We further multiply $\Delta w_i = w_i \cdot 0.01$ by n_{bad_i}/k_{reco}. Therefore, a weight with greater n_{bad_i} (a less informative weight) will be reduced more.

- We also decrease Δw_i if P_{train} is already well classified. N_{bad} is defined as:

  $$N_{bad} = |\{P' \in S_{bad} | d(P_{train}, P') \leq d_{maxgood}\}|$$

  Specifically, we multiply Δw_i by $0.2 + N_{bad}/k_{reco}$. N_{bad} can be considered an overall measure of how poorly the current point is classified, such that points which are already well-classified in each iteration have less of an impact on the weights. The addition of 0.2 indicates that even perfectly classified points still have some small impact on the weights (so that the weight adjustment will not nullify their perfect classification).

Both of these above changes improved our classification accuracy. We achieved our best results with $k_{reco} = 5$.

We initialized the weight vector W randomly by choosing a random value for each w_i uniformly between 0.5 and 1.5. Adding randomness gave us a chance of finding better solutions than a deterministic algorithm as we could avoid local maxima that bind our classifier away from the global maximum.

Note that we are not claiming these particular choices of parameters and constants yield an optimal attack, and further work may yet uncover improved attacks against defenses without provable privacy guarantees.

5 Attack evaluation

Our attack is specifically designed to find gaps in defenses, and in this section we will demonstrate its efficacy with experimentation on real web traffic. We will first begin by showing the effectiveness of our scheme against Tor with its default packet padding and order randomization defense in Section 5.1. This setting is a good standard basis of comparison as WF is a threat to the privacy guarantees provided by Tor, and several of the latest state-of-the-art attacks are designed for and evaluated on Tor. We will see that our attack performs better than the best known attacks. The parameters of our attack can be modified to decrease the false positive rate at the cost of decreasing the true positive rate, and we examine the tradeoff in Section 5.2. Then, we show that our attack is also more powerful than known attacks on various known and published defenses in Section 5.3, with a number of defenses shown to be nearly completely ineffective against our scheme.

5.1 Attack on Tor

We validate our attack in two experimental settings to demonstrate the effectiveness of our attack on Tor.

First, we perform experiments in an open-world experimental setting. Even though the number of pages in the world wide web is far too large for us to train on, we can achieve realistic results by limiting the objective of the attacker. Here, the attacker wants to decide whether or not a packet sequence comes from a monitored page; additionally, for monitored pages, the attacker aims to identify the page. We denote the non-monitored page set

that the attacker uses for training as C_0, and the effects of varying its size will be tested in evaluation. This open-world experimental setting gives us realistic results for plausible attackers.

We use a list of 90 instances each of 100 sensitive pages as well as 1 instance each of 5,000 non-monitored pages. We note that this problem is more difficult for the attacker than any that has been evaluated in the field, as other authors have evaluated their schemes on either strictly closed-world settings or very small open-world problems (a few monitored pages). It is a realistic goal for the attacker to monitor a large set of pages in the open-world setting.

Our list of 100 monitored pages was compiled from a list of blocked web pages from China, the UK, and Saudi Arabia. These include pages ranging from adult content, torrent trackers, and social media to sensitive religious and political topics. We selected our list of 5,000 non-monitored pages from Alexa's top 10,000 [1], in order, excluding pages that are in the list of monitored pages by domain name. The inherent differences between the above data sets used for training and testing assist classification, just as they would for a realistic attacker. Page loading was done with regular circuit resetting, no caches and time gaps between multiple loads of the same page (as suggested by Wang and Goldberg [23]), such that the attacker will not use the same circuits as the target client, or collect its data at the same time. We used iMacros 8.6.0 on Tor Browser 3.5.1 to collect our data.

Training the k-Nearest Neighbour classifier is required to learn the correct weights. We learn the weights by splitting part of the training set for weight adjustment and evaluation as above. We perform weight adjustment for $R = 6000$ rounds on 100 pages and 60 instances each, which means that every instance is cycled over once. Then, accuracy is computed over the remaining 30 instances each, on which we perform all-but-one cross validation. The use of cross validation implies that the attacker will never train on the same non-monitored pages that the client visits.

For our attack, we decided that a point should be classified as a monitored page only if all k neighbours agree on which page it is, and otherwise it will be classified as a non-monitored page. This helped reduce false positives at a relatively small cost to the true positives. We vary the number of neighbours k from 1 to 15 as well as the number of non-monitored training pages $|C_0|$ used from 10 to 5000,[5] and we show our results in Figure 1. We measure the True Positive Rate (TPR), which is the probability that a monitored page is correctly classified as that particular monitored page, and the False Positive Rate (FPR), which is the probability that a non-monitored page is incorrectly identified as being monitored.[6] We can achieve TPR 0.85 ± 0.04 for FPR 0.006 ± 0.004, or respectively TPR 0.76 ± 0.06 for FPR 0.001 ± 0.001.

[5] We note that the choice of $|C_0|$ does *not* represent a world with fewer pages available to the client—it is the attacker's decision on how much he wishes the bias towards non-monitored sites to be. The visited sites are always drawn from Alexa's top 10,000.

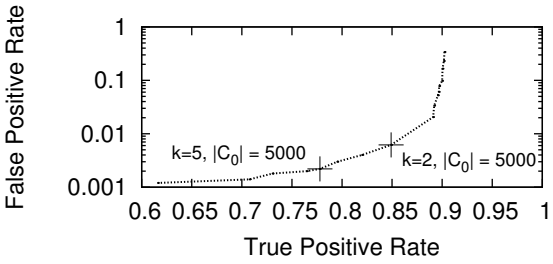

Figure 1: Performance of our attack while varying the attack parameters k and $|C_0|$. Only the y-axis is logarithmically scaled.

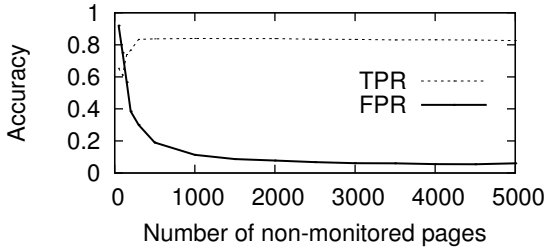

Figure 2: True Positive Rate and False Positive Rate changes in OSAD [23] as the open set size increases. There is almost no change in either value after $|C_0| > 2500$.

We compare these values to OSAD, which we apply to our data set as well, and show the results in Figure 2. Increasing the number of non-monitored pages $|C_0|$ increases TPR and reduces FPR. After $|C_0| > 2500$, we could not see a significant benefit in adding more elements. At $|C_0| = 5000$, the classifier achieves a TPR of 0.83 ± 0.03 and a FPR of 0.06 ± 0.02.

We see that OSAD cannot achieve FPR values nearly as low as ours, and it may be considered impractical for the attacker to monitor large sets in the open-world setting with the old classifier, especially if the base incidence rate is low. For example, if the base incidence rate of the whole sensitive set is 0.01 (99% of the time the client is visiting none of these pages), and our new classifier claims to have found a sensitive site, the decision is correct at least 80% of the time, the rest being false positives. For Wang and Goldberg's classifier, the same value would be about 12%. The difference is further exacerbated with a lower base incidence rate, which may be realistic for particularly sensitive web sites.

[6] If a monitored page is incorrectly classified as a *different* monitored page or as a non-monitored page, it is a false negative.

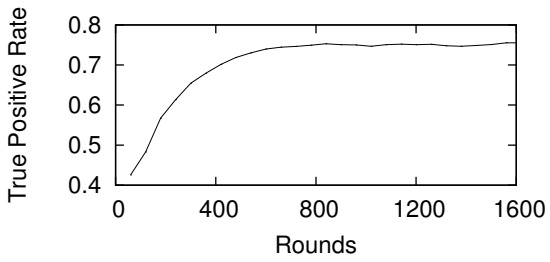

Figure 3: TPR when varying the number of rounds used for training our attack, with $k = 5$ and $|C_0| = 500$. FPR is not shown because there is very little change over time.

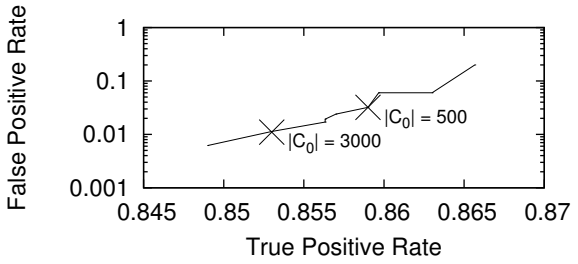

Figure 4: Results for FPR vs. TPR while varying bias towards non-monitored data set C_0. $k = 2$.

The training and testing time for our classifier (weight adjustment) is very small compared to previous state-of-the-art classifiers. The number of rounds, R, determines the quality of our weights. We show in Figure 3 how the true positive rate changes with R on $|C_0| = 500$ non-monitored sites and $k = 5$ neighbours. We see that the accuracy levels off at around 800 rounds, and did not drop up to 30,000 rounds.

The weight training time scales linearly with R and also scales linearly with the number of instances used for weight training. The training time is around $8 \cdot 10^{-6} \cdot |S_{train}| \cdot R$ CPU seconds, measured using a computing cluster with AMD Opteron 2.2 GHz cores. This amounts to around 120 CPU seconds for 1000 rounds in our set with $|C_0| = 5000$. This can be compared to around 1600 CPU hours on the same data set using OSAD and 500 CPU hours using that of Cai et al. Training time also scales quadratically with the number of training instances with these previous classifiers.

The testing time amounts to around 0.1 CPU seconds to classify one instance for our classifier and around 800 CPU seconds for OSAD, and 450 CPU seconds for Cai et al. The testing time per instance scales linearly with the number of training elements for all three classifiers. We can reduce the training and testing time for our classifier further by around 4 times if we remove the unique packet length features, which are useless for Tor cells.

We also perform experiments on the closed-world experimental setting. Under the closed-world experimental setting, the client does not visit non-monitored pages. We use the same data set of sensitive pages as above

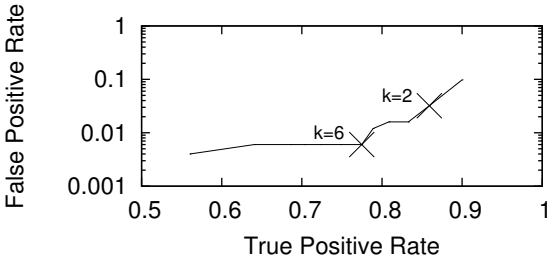

Figure 5: Best results for FPR vs. TPR while varying number of neighbours k. $|C_0| = 500$.

for these experiments. Although the closed-world setting does not carry the same realistic implications as the open-world setting, it focuses attention on the ability of the classifier to distinguish between pages and it has been a useful basis of comparison in the field. We also tested our classifier on the data set used by Wang and Goldberg to facilitate a more direct comparison, and the accuracy was 0.95 ± 0.02 compared to 0.91 ± 0.06 for OSAD and 0.88 ± 0.03 for Cai et al. We also compared them on our new data set, and the accuracy was 0.91 ± 0.03 for ours and 0.90 ± 0.02 for OSAD. There appears to be no significant difference in the closed-world scenario, although the superior accuracy of our classifier under the realistic open-world scenario is clear.

5.2 Training confidence

The numbers for true and false positive rates as shown above may not be desirable for some cases. The optimal numbers depend on the expected base rate of the monitored activity as well as the application intended by the attacker. Parameters of our attack can be adjusted to increase true positive rate at the cost of increasing false positive rate, or vice versa.

We can vary the size of the non-monitored training page set to affect accuracy as our implementation of the k-Nearest Neighbour classifier is susceptible to bias towards larger classes. We fix the number of neighbours at $k = 2$, vary the number of non-monitored training pages $|C_0|$ from 10 to 5000 and show the results in Figure 4.

We can also vary k, the number of neighbours. We fix the number of non-monitored pages, $|C_0|$, at 500, and vary k from 1 to 15, showing the results in Figure 5. Decreasing $|C_0|$ and decreasing k each increases both true positives and false positives.

We can see that varying the number of neighbours used is much more important for determining TPR than varying the size of C_0, the set of non-monitored pages. In fact, almost all of the graph in Figure 1 can be drawn only by varying k with $|C_0| = 5000$, suggesting that it is advantageous for the attacker to have a large number of non-monitored training pages.

Table 1: Accuracy of our attack on various defenses. Closed-world simulation is used to enable comparison with previous known results.

Defense	Accuracy	Bandwidth overhead
Traffic morphing [24]	0.82 ± 0.06	$50\% \pm 10\%$
HTTPOS split [16]	0.86 ± 0.03	$5.0\% \pm 0.6\%$
Decoy pages [17]	0.30 ± 0.06	$130\% \pm 20\%$
BuFLO [6]	0.10 ± 0.03	$190\% \pm 20\%$

5.3 Attack on Other Defenses

Our attack is specifically designed to break WF defenses that leave features open for classification. The analysis in previous sections was performed on Tor packets, which already uses padding, pipelining and order randomization. We add further defenses on top of Tor's defenses. The list of defenses we evaluate in this section are as follows:

- Traffic morphing [24]. Traffic morphing maps packet sizes from one site to a packet distribution drawn from another site, in an attempt to mimic the destination site. In our implementation, each site attempted to mimic `google.com` as it is reasonable to assume that the client wishes to mimic the most popular page.
- HTTPOS split [16]. Although HTTPOS has a large number of features, one of its core features is a random split on unique packet lengths by cleverly utilizing HTTP range requests. We analyze HTTPOS by splitting incoming packets and also padding outgoing packets.[7]
- Panchenko's decoy pages [17]. As a defense against their own attack, Panchenko et al. suggested that each real page should be loaded with a decoy page. We chose non-monitored pages randomly as decoy pages.
- BuFLO [6]. Maximum size packets are sent in both directions at equal, constant rates until the data has been sent, or until 10 seconds have passed, whichever is longer.

We implement these defenses as simulations. For Panchenko's noise and BuFLO we implement them using Tor cells as a basic unit in order to reduce unnecessary overhead from these defenses when applied on Tor. We assume that the attacker is aware of these defenses and collects training instances on which the defense is applied; this is realistic as the above defenses are all distinctive and identifiable.

We apply our attack, and show the results in Table 1. This can be compared to a minimum accuracy of 0.01 for random guessing. We see that even with large overhead, the defenses often fail to cover the page, and our attack always performs significantly better than random guessing. For BuFLO, our particular data set gave a larger overhead than previous work [22] because most packet sequences could be loaded within 10 seconds and therefore required end-of-sequence padding to 10 seconds. In particular, traffic morphing and HTTPOS split have almost no effect on the accuracy of our attack.

6 Defense

In this section, we design a provably private defense—a defense for which there exists an upper bound on the accuracy of any attack (given the data set). As Tor is bandwidth-starved [21], we attempt to give such a defense with the minimum bandwidth cost. This is an extension of the idea proposed by Wang and Goldberg [22] for their defense, Tamaraw.

In Section 6.1, we first show how such an upper bound can be given for *simulatable, deterministic defenses*—that is, this class of defenses can be made to be provably private. We then show in Section 6.2 that the optimal defense strategy (lowest bandwidth cost) in such a class is to compute supersequences over sets of packet sequences (anonymity sets). We try to approximate the optimal defense strategy, by describing how these sets can be chosen in Section 6.3, and how the supersequence can be estimated in Section 6.4.

6.1 Attacker's upper bound

We describe how we can obtain an upper bound on the accuracy of any attack given a defended data set. The attacker, given an observation (packet sequence) p, wishes to find the class it belonged to, $C(p)$.

To calculate the maximum success probability given the testing set, we assume the greatest possible advantage for the attacker. This is where the attacker is allowed to train on the testing set.[8] In this case the attacker's optimal classification strategy is to record the true class of each observation, $(p, C(p))$. The attacker will only ever make an error if the same observation is mapped to several different classes, which are indistinguishable for the observation. We denote the possibility set of p as the multiset of classes with the same observation p, $Q(p) =$

[7] HTTPOS has been significantly modified by its authors since their original publication, in part due to the fact that Cai et al. were able to break it easily [4].

[8] Our testing set is in fact a multiset as repeated observation-class pairs are possible.

$\{C_1, C_2, \ldots\}$ ($C(p) \in Q(p)$), where the occurrence count of a class is the same as in the testing set with observation p.

The attacker's optimal strategy is to find the class C_{max} that occurs the most frequently for the same observation p, and during classification the attacker will return C_{max} for the observation p. This will induce an accuracy value upon p:

$$Acc(p) = \frac{|\{C \in Q(p) | C = C_{max}\}|}{|Q(p)|}$$

This method returns the best possible accuracy for a given testing set as it makes the lowest possible error for observations mapping to multiple classes.

Cai et al. [3] have proposed two different ways to denote the overall accuracy of a set of packet sequences:

- Non-uniform accuracy. This is the mean of accuracies $Acc(p)$ for $p \in S_{test}$.
- Uniform accuracy. This is the maximum accuracy $Acc(p)$ for $p \in S_{test}$

Tamaraw can only achieve non-uniform accuracy. In this work, we design a defense for uniform accuracy, but the defense can be extended to other notions as well. While we will use different sets to train our defense and test it on client behaviour, we will say that the defense has a maximum uniform accuracy as long as it does so on the training set (as it is always possible to construct a testing set on simulatable, deterministic defenses on which at least one page has an accuracy of 1). A defense that achieves a maximum uniform accuracy of A_u automatically does so for non-uniform accuracy, but not vice-versa. In the following we work with a uniform prior on a fixed-size testing set to facilitate comparison with previous work.

6.2 Optimal defense

In this section, we show the bandwidth-optimal simulatable, deterministic defense. As we work with Tor cells, in the following a packet sequence can be considered a sequence of -1's and 1's (downstream and upstream packets respectively), which is useful for hiding unique packet lengths [22]. We say that sequence q is a subsequence of sequence p (or that p is a supersequence of q) if there exists a set of deletions of -1 and 1 in p to make them equal (maintaining order). With abuse of notation, we say that if S is the input packet sequence multiset, then $D(S) = \{D(p) | p \in S\}$ denotes the output packet sequence multiset after application of the defense. The cost (bandwidth overhead) of $D(p)$ is $B(D(p)) = \frac{|D(p)| - |p|}{|p|}$, and similarly for a set of packet sequences the overhead is $B(D(S)) = \frac{\sum_{p \in S} |D(p)| - \sum_{p \in S} |p|}{\sum_{p \in S} |p|}$. Given S, we want to identify D such that $B(D(S))$ is minimal.

For each packet sequence p_1, let us consider the set of packet sequences that map to the same observation after the defense is applied, which we call the *anonymity set* of p_1. We write the set as $A(p_1) = \{p_1, p_2, \ldots, p_E\}$; i.e. $D(p_1) = D(p_i)$ for each i. The shortest $D(p_1)$ that satisfies the above condition is in fact the shortest common supersequence, written as $f_{scs}(A(p_1)) = D(p_i)$ for each $1 \leq i \leq E$.

In other words, the optimal solution is to apply the shortest common supersequence function to anonymity sets of input sequences. This defense can be applied with the cooperation of a proxy on the other side of the adversary; on Tor, for example, this could be the exit node of the circuit. However, finding such an optimal solution requires solving two hard problems.

Anonymity set selection. First, given the set of all possible packet sequences, we want to group them into anonymity sets such that, for a given bound on attacker accuracy, the overhead will be minimized.

The shortest common supersequence (SCS) problem. Then, we must determine the SCS of all the packet sequences in the anonymity set. This is in general NP-hard. [13]

In the next two sections we describe our solutions to the above problems.

6.3 Anonymity set selection

We note that the client is not always able to choose anonymity sets freely. For example, the client cannot easily know which anonymity set a page load should belong to before seeing the packet sequence. While the client can gain information that assists in making this decision (the URL, previous page load data, training data, information about the client network, the first few packets of the sequence, etc.), the mere storage and usage of this information carries additional privacy risks. In particular, the Tor Browser keeps no disk storage (including no cache except from memory), so that storing extraneous information puts the client at additional risk. In this section, we describe how realistic conditions impose restrictions on the power of the client to choose anonymity sets.

We formalize this observation by imposing additional limits on anonymity set selection in the defense D trained on testing set S_{test}. We define four levels of information for a client applying a simulatable, deterministic website fingerprinting defense:

Table 2: Relationship between different levels of information and how we train and test our supersequences. Under "Supersequence", we describe what supersequences we would use at this level of information. Clustering is done if we want multiple supersequences.

Information	Supersequence	Training and Testing
No information	One supersequence	Different sites, instances
Sequence end information	One supersequence, stopping points	Different sites, instances
Class information	Multiple supersequences, stopping points	Same sites, different instances
Full information	Multiple supersequences	Same sites, instances

1. No information. The client has no information at all about the packet sequence to be loaded. This means $A(p) = S_{test}$, that is to say all sequences map to a single anonymity set.

2. Sequence end information. The client knows when the sequence has ended, but this is the only information the client gets about the packet sequence. This means that D can only vary in length; for any p, q, such that $|D(p)| \geq |D(q)|$, then the first $|D(q)|$ packets of $D(p)$ are exactly $D(q)$, that is, we say that $D(q)$ is a prefix of $D(p)$.

3. Class-specific information. Only the identity of the page is known to the client, and the client has loaded the page before with some information about the page, possibly with realistic offline training. The client cannot distinguish between different packet sequences of the same page (even though the page may be multi-modal). This is the same as the above restriction but only applied if p and q are packet sequences from the same web page.

4. Full information. No restrictions are added to D. The client has prescient information of the full packet sequence. Beyond class-specific information, the client can gain further information by looking into the future at the contents of the packet sequence, learning about her network, and possibly using other types of extraneous information. This level is not generally of practical interest except for serving as a bound for any realistic defense.

We use clustering, an unsupervised machine learning technique, to find our anonymity sets. We show how the above levels of information affect how supersequences will be computed and how testing needs to be performed in Table 2.

Optimality under the above levels of information requires the computation of supersequences over anonymity sets. If we have only sequence end information, there is only one supersequence, and we do not need to perform clustering. Instead, possible outputs of the defense simply correspond to a prefix of the one supersequence, terminating at one of a specified set of stopping points. We find the stopping points by selecting the earliest points where our maximum uniform accuracy would be satisfied. All packet sequences sent under this defense will be padded to the next stopping point. This is similar to a strategy suggested by Cai et al. [3]

If we have class-level information, we need to perform two levels of anonymity set selection. On the first level, we cluster the packet sequences within each class to decide which supersequence the client should use. For this level of clustering, we first decide on the number of supersequences in the set. Then, we randomly choose a number of "roots" equal to this number of supersequences. We cycle over every root, assigning the closest packet sequence that has not yet been classified. For this we need to define a distance between each pair of packet sequences p and q. Suppose p' and q' are the first $min(|p|,|q|)$ packets of p and q respectively. The distance between p and q is given as $2|f_{scs}(p',q')| - |p'| - |q'|$. We use this distance to measure how different two packet sequences are, without considering their respective lengths, which would be addressed by the second level. On the second level, we find stopping points, with the same strategy as that used under sequence end information. The use of an additional first level of clustering reduces the number of stopping points available for use, given a fixed number of clusters, so that using too many clusters may in fact have a higher bandwidth overhead (see Section 7).

For full information, we perform clustering with the distance between two packet sequences p and q as $2|f_{scs}(p,q)| - |p| - |q|$. Here we select roots with evenly spread out lengths.

6.4 SCS approximation

For the SCS of two packet sequences there is an exact solution that can be found using dynamic programming; however, the SCS of multiple sequences is in general NP-hard [13].

We present a simple algorithm that approximates a solution to the shortest common supersequence problem. To approximate $f_{scs}(\{p_1, p_2, \ldots, p_n\})$, we define a counter for each packet sequence $c_1, c_2, \ldots, c_n$, which starts at 1. We count the number of sequences for which

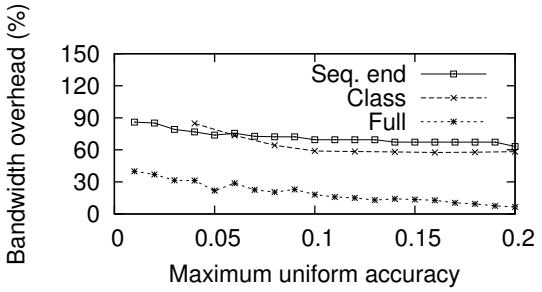

Figure 6: Bandwidth overhead for three levels of information: sequence end information (Seq. end), class-specific information (Class), and full information (Full). Using no information results in a bandwidth overhead that is much higher than that shown in the graph.

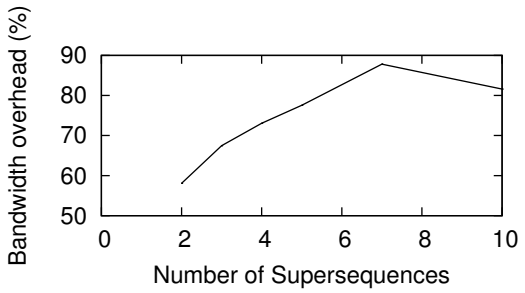

Figure 7: Bandwidth overhead for class-specific information if more than two supersequences are used, at 20 clusters. The number of stopping points available decreases, but the total number of supersequences times the number of stopping points is always at least 20.

the c_i-th element of p_i is an outgoing packet. If the number exceeds $n/4$, we append an outgoing packet to the common supersequence, and increment all c_i for which the c_i-th element of p_i is an outgoing packet by 1. Else, we append an incoming packet, and increase the corresponding counts by 1. We do this iteratively until each counter c_i becomes $|p_i|+1$, and the algorithm terminates. The choice of $n/4$ is because for web page loading, there are fewer outgoing packets than incoming packets, and this choice reduces our overhead significantly.

We note that it is easy to construct cases where the above algorithm performs very poorly. In fact, it is known that any polynomial-time approximation algorithm of shortest common supersequences cannot have bounded error [13].

7 Defense evaluation

In this section we evaluate our defense for bandwidth overhead, as well as its effectiveness in stopping our new attack.

We implemented our defenses with different levels of information as seen above. We used the same data set used to test our attacks—100 sites, 30 instances each—and attempted to protect them. The defender attempts to achieve a given maximum uniform accuracy (by determining the number of clusters or stopping points). We show the results in Figure 6. For class-level information, we used two supersequences and $N/2$ stopping points in each supersequence. We can see the full information setting has a much lower bandwidth overhead than sequence end information or class-level information. With our clustering strategy, using two supersequences under class-level information is only sometimes beneficial for the overhead. It is possible that a clever clustering strategy for class-level information could achieve lower bandwidth overheads.

For class-level information, we used two supersequences as above. It is interesting to know if increasing the number of supersequences (and correspondingly lowering the number of stopping points) will give better bandwidth overhead. In other words, we want to know if it is worth suffering greater overhead for padding to stopping points to have more finely tuned supersequences. We fix the target maximum uniform accuracy to 20%. The results are shown in Figure 7. We can see that using more than two supersequences only increases the bandwidth overhead. It is possible that if the defender can tolerate a higher maximum uniform accuracy, then it would be optimal to use more than two supersequences.

Finally, we apply our new attack to a class-level defense with a maximum uniform accuracy of 0.1, where the overhead is approximately $59\% \pm 3\%$. We achieved an accuracy of 0.068 ± 0.007. This can be compared to Table 1, where we can see that the attack achieved an accuracy of 0.30 ± 0.06 for Panchenko's decoy pages with an overhead of $130\% \pm 20\%$ and an accuracy of 0.10 ± 0.03 for BuFLO with an overhead of $190\% \pm 20\%$. Furthermore, we do not know if there exist better attacks for these defenses, but we know that no attack can achieve a better accuracy than 0.1 on our defense (using the same data set). We also compared our work with Tamaraw, which had a $96\% \pm 9\%$ overhead on the same data set for non-uniform accuracy. Our attack achieved an accuracy of 0.09 ± 0.02, although highly non-uniformly. Indeed, on 16 sites out of 100, the accuracy of the attacker was more than 0.2, and the most accurately classified site had accuracy 0.6.

8 Discussion

8.1 Realistically applying an attack

Like other website fingerprinting works in the field, we make the assumption that the attacker has an oracle that

can answer whether or not a particular sequence is generated from a single page load, and that the user does not prematurely halt the page load or perform other types of web activity. Here we discuss a few strategies to deal with possible sources of noise when applying website fingerprinting to the real world.

The attacker can use a number of signals to identify the start of a packet sequence. We found that the start of a packet sequence generally contains around three times more outgoing packets than the rest of the sequence. If the user is accessing a page for which she does not have a current connection (i.e. most likely the user is visiting a page from another domain), then the user will always send one or two outgoing connections (depending on the browser setting) to the server, followed by acceptance from the server, followed by a GET request from the main page, and then by data from the server. This particular sequence could be identifiable.

Unfortunately for Tor users, website fingerprinting is made easier due to a number of design decisions. On Tor, users are discouraged from loading videos, using torrents, and downloading large files over Tor, which are types of noise that would interfere with website fingerprinting. It is hard to change user settings on the Tor Browser; the configuration file is reset every time the Tor Browser is restarted, which implies that different Tor users have similar browser settings. As there is no disk caching, Tor users have to log in every time the Tor Browser is restarted before seeing personalized pages. For example, Facebook users on Tor must go through the front page, which has no variation and is easily identifiable. This is meant to preserve privacy from server-side browser fingerprinting attacks, but they also make website fingerprinting easier.

8.2 Realistic consequences of an attack

Here we discuss how our attack can be used realistically to break the privacy of web users. Our attack is not all-powerful; it is not likely to find a single sensitive page access among millions without error. The quality of the results depends on the base incidence rate of the client's access. With our classifier, if an attacker wishes to identify exactly which of a set of 100 pages a client is visiting, and she almost never visits those pages (less than 0.1% of page visits), then false alarms will overwhelm the number of true positives. We note that many sensitive pages have high rates of incidence as they are within Alexa's top 100 (torrent sites, adult sites, social media), especially if the client feels it necessary to use Tor.

We envision our attack as a strong source of information that becomes more powerful with the use of other orthogonal sources of information. For instance, a government agency observes that a whistleblower has released information on a web page, or that she has just posted a sensitive or incendiary article on a blog, and it is known that this whistleblower is likely to use Tor. The agency will only need to search amongst Tor streams in the last few minutes within the nation (or a smaller local area). As Tor streams are easily identifiable [7], the number of Tor users at any given moment is small enough for our accurate attack to lead to the capture of a Tor-using dissident. This strongly suggests that some sort of defense is necessary to protect the privacy of web clients.

8.3 Reproducibility of our results

To ensure reproducibility and scientific correctness, we publish the following: [9]

- The code for our new attack. This includes our feature set, parameters used for our weight learning process, and a number of weight vectors we learned which succeeded at classification against specific defenses, including the Tor data set.
- The code for our new defense. This includes the clustering strategy and the computation for stop points, as well as the supersequences we eventually used to achieve the results in this paper.
- Our implementations of known attacks and defenses, which we compared and evaluated against ours.
- The data sets we used for evaluation. This includes the list of monitored and non-monitored sites we visited over Tor, and the TCP packets we collected while visiting those sites and which we processed into Tor cells. We also include the feature vectors we computed over this data set.

9 Conclusion

In this work, we have shown that using an attack which exploits the multi-modal property of web pages with the k-Nearest Neighbour classifier gives us a much higher accuracy than previous work. We use a large feature set and learn feature weights by adjusting them based on shortening the distance towards points in the same class, and we show that our procedure is robust. The k-NN costs only seconds to train on a large database, compared to hundreds of hours for previous state-of-the-art attacks. The attack further performs well in the open-world experiments if the attacker chooses k and the bias towards non-monitored pages properly. Furthermore, as

[9] They can be found at `https://crysp.uwaterloo.ca/software/webfingerprint/`

the attack is designed to automatically converge on unprotected features, we have shown that our attack is powerful against all known defenses.

This indicates that we need a strong, provable defense to protect ourselves against ever-improving attacks in the field. We identify that the optimal simulatable, deterministic defense is one with supersequences computed over the correct anonymity sets. We show how to construct a class of such defenses based on how much information the defender is expected to have, and we evaluate these defenses based on approximations over supersequence computation and anonymity set selection. We show a significantly improved overhead over previous simulatable, deterministic defenses such as BuFLO and Tamaraw at the same security level.

Acknowledgements We would like to thank the anonymous reviewers for their suggestions. This research was funded by NSERC, ORF, and The Tor Project, Inc. This work was made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET: `www.sharcnet.ca`) and Compute/Calcul Canada.

References

[1] Alexa — The Web Information Company. www.alexa.com.

[2] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine. Privacy Vulnerabilities in Encrypted HTTP Streams. In *Privacy Enhancing Technologies*, pages 1–11. Springer, 2006.

[3] X. Cai, R. Nithyanand, and R. Johnson. New Approaches to Website Fingerprinting Defenses. *arXiv*, abs/1401.6022, 2014.

[4] X. Cai, X. Zhang, B. Joshi, and R. Johnson. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *Proceedings of the 19th ACM Conference on Computer and Communications Security*, pages 605–616, 2012.

[5] H. Cheng and R. Avnur. Traffic Analysis of SSL-Encrypted Web Browsing. `http://www.cs.berkeley.edu/~daw/teaching/cs261-f98/projects/final-reports/ronathan-heyning.ps`.

[6] K. Dyer, S. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, pages 332–346, 2012.

[7] D. Fifield, N. Hardison, J. Ellithorpe, E. Stark, D. Boneh, R. Dingledine, and P. Porras. Evading censorship with browser-based proxies. In *Privacy Enhancing Technologies*, pages 239–258, 2012.

[8] Y. Gilad and A. Herzberg. Spying in the Dark: TCP and Tor Traffic Analysis. In *Privacy Enhancing Technologies*, pages 100–119. Springer, 2012.

[9] X. Gong, N. Kiyavash, and N. Borisov. Fingerprinting Websites using Remote Traffic Analysis. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, pages 684–686. ACM, 2010.

[10] D. Herrmann, R. Wendolsky, and H. Federrath. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, pages 31–42, 2009.

[11] A. Hintz. Fingerprinting Websites Using Traffic Analysis. In *Privacy Enhancing Technologies*, pages 171–178. Springer, 2003.

[12] S. Jana and V. Shmatikov. Memento: Learning secrets from process footprints. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, pages 143–157. IEEE, 2012.

[13] T. Jiang and M. Li. On the approximation of shortest common supersequences and longest common subsequences. *SIAM Journal on Computing*, 24(5):1122–1139, 1995.

[14] M. Liberatore and B. Levine. Inferring the Source of Encrypted HTTP Connections. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 255–263, 2006.

[15] L. Lu, E.-C. Chang, and M. C. Chan. Website Fingerprinting and Identification Using Ordered Feature Sequences. In *Computer Security–ESORICS 2010*, pages 199–214. Springer, 2010.

[16] X. Luo, P. Zhou, E. W. Chan, W. Lee, R. K. Chang, and R. Perdisci. HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. In *Proceedings of the 18th Network and Distributed Security Symposium*, 2011.

[17] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *Proceedings of the 10th ACM Workshop on Privacy in the Electronic Society*, pages 103–114, 2011.

[18] M. Perry. Experimental Defense for Website Traffic Fingerprinting. https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting, September 2011. Accessed Feb. 2014.

[19] M. Perry. A Critique of Website Traffic Fingerprinting Attacks. https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks, November 2013. Accessed Feb. 2014.

[20] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. Statistical Identification of Encrypted Web Browsing Traffic. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 19–30. IEEE, 2002.

[21] Tor. Tor Metrics Portal. https://metrics.torproject.org/. Accessed Oct. 2013.

[22] T. Wang and I. Goldberg. Comparing website fingerprinting attacks and defenses. Technical Report 2013-30, CACR, 2013. http://cacr.uwaterloo.ca/techreports/2013/cacr2013-30.pdf.

[23] T. Wang and I. Goldberg. Improved Website Fingerprinting on Tor. In *Proceedings of the 12th ACM Workshop on Privacy in the Electronic Society*, 2013.

[24] C. Wright, S. Coull, and F. Monrose. Traffic Morphing: An Efficient Defense against Statistical Traffic Analysis. In *Proceedings of the 16th Network and Distributed Security Symposium*, pages 237–250, 2009.