DE GRUYTER OPEN Proceedings on Privacy Enhancing Technologies ; 2016 (4):21–36

Tao Wang* and Ian Goldberg

On Realistically Attacking Tor with Website Fingerprinting

Abstract: Website fingerprinting allows a local, passive observer monitoring a web-browsing client's encrypted channel to determine her web activity. Previous attacks have shown that website fingerprinting could be a threat to anonymity networks such as Tor under laboratory conditions. However, there are significant differences between laboratory conditions and realistic conditions. First, in laboratory tests we collect the training data set together with the testing data set, so the training data set is fresh, but an attacker may not be able to maintain a fresh data set. Second, laboratory packet sequences correspond to a single page each, but for realistic packet sequences the split between pages is not obvious. Third, packet sequences may include background noise from other types of web traffic. These differences adversely affect website fingerprinting under realistic conditions. In this paper, we tackle these three problems to bridge the gap between laboratory and realistic conditions for website fingerprinting. We show that we can maintain a fresh training set with minimal resources. We demonstrate several classification-based techniques that allow us to split full packet sequences effectively into sequences corresponding to a single page each. We describe several new algorithms for tackling background noise. With our techniques, we are able to build the first website fingerprinting system that can operate directly on packet sequences collected in the wild.

DOI 10.1515/popets-2016-0027
Received 2016-02-29; revised 2016-06-02; accepted 2016-06-02.

1 Introduction

In 2009, Panchenko et al. [20] introduced a website fingerprinting (WF) attack that successfully achieved accurate web page classification on Tor. WF threatens clients seeking to hide their online behaviour from *local* adversaries—ones able to monitor the network close to the client, such as ISPs, wiretappers, and packet sniffers. Since then, researchers have published more accurate attacks, improving the true positive rate (TPR) [3, 25] and cutting down the false positive rate (FPR) [24] to practical levels (far below 1%). Researchers have also applied WF techniques to circuit fingerprinting, allowing adversaries to discover and identify Tor hidden services [14]. They have shown that these attacks are computationally cheap and effective in the open-world setting [24]. However, some researchers remain unconvinced that these attacks are effective in the wild [13, 21].

Indeed, the attacks have not been demonstrated to be effective in the wild; they were proven only under laboratory conditions. Recently, Juarez et al. [13] identified significant differences between attacks in the wild and attacks proven under laboratory conditions. They noted that previous works on WF attacks made five limiting assumptions:[1]

1. Closed world: Under the closed-world model, the WF attack is never tested with web pages outside a fixed set of monitored pages. A WF attack that operates under the open-world model must be able to determine whether or not a web page is in the set of monitored pages.
2. Replicability: The attacker's classification training set is collected under the same conditions as the client. Specifically, a stale training set can cause WF accuracy to deteriorate.
3. Browsing behaviour: Users browse the web sequentially, one page after the other.
4. Page load parsing: The adversary knows when pages start and end. For example (related to the above), most users may have significant time gaps between page loads.
5. No background traffic: The adversary can filter out all background traffic.

Assumption 1 has been dealt with by previous work [20, 24, 25]. For example, the kNN attack by Wang et al. [24] can achieve a true positive rate of 85% and a false positive rate of 0.6% in the open-world model, with no limit on the number of web pages. In this work, we tackle assumptions 2 to 5, as follows:

Freshness (assumption 2). We determine empirically that the attacker needs only a small amount of data to perform WF effectively, and therefore it is easy to keep it fresh.

***Corresponding Author: Tao Wang:** Hong Kong University of Science and Technology, E-mail: taow@cse.ust.hk
Ian Goldberg: University of Waterloo, E-mail: iang@cs.uwaterloo.ca

Tao Wang did most of his work while at the University of Waterloo.

[1] In their original work, they listed six assumptions, of which we omitted the first because it only concerns one specific attack by Cai et al. [3] rather than website fingerprinting attacks in general.

BY-NC-ND
Unauthenticated
Download Date | 10/23/16 6:26 PM

Replicability is indeed possible. In addition, we propose and test several schemes for updating the training set.

Splitting (assumptions 3 and 4). We show that it is indeed possible for adversaries to know when pages start and end from full realistic packet sequences, even if the user is visiting multiple pages at once. We turn realistic packet sequences into laboratory packet sequences by *splitting*: distinguishing between different web pages that may occur sequentially or even in parallel. We demonstrate the effectiveness of time-based splitting and classification-based splitting.

Background noise (assumption 5). We show several new algorithms to deal with background noise, and we demonstrate that an attacker who adjusts his training set can still perform website fingerprinting accurately on traffic with two types of high-bandwidth noise: audio streaming noise and file download noise.

We emphasize that this work does *not* propose a new classifier to improve the classification accuracy of known WF attacks under laboratory conditions; rather, we augment any WF attack with tools to operate under realistic conditions. Furthermore, we do not know the final accuracy of website fingerprinting in the wild (and thus whether or not this attack is truly practical), because this depends significantly on user behaviour for which we have limited information. Nevertheless we will evaluate each component of our system individually to show that we have made WF attacks more realistic. The code and data for our system is available for download at https://crysp.uwaterloo.ca/software/webfingerprint/.

Our results are presented as follows. In Section 2 we describe the related work that led to WF approaching practicality and thus motivated this work. In Section 3 we give the background and terminology of this paper. In Section 4 we demonstrate that an attacker can practically maintain a fresh training set. In Section 5 we describe how we solve the splitting problem, and we present the results in Section 6. We tackle noise removal in Section 7. Then in Section 8 we discuss the reproducibility of our work, and we conclude in Section 9.

2 Related Work

In this section, we describe how WF progressed from a theoretical attack under specific situations towards a practical threat for anonymous communications in general. This section will focus on the practicality of the schemes; we refer the reader to previous work [1–3, 5, 8, 11, 12, 15, 17, 20, 22, 24, 25] for a more technical description of the particular machine-learning classifiers they used.

A practical WF attacker uses the open-world model, where the attacker chooses a set of *monitored pages* for the classifier. When the client visits any of the monitored pages, the classifier attempts to identify the page. All other web pages are non-monitored and the classifier only needs to identify that non-monitored pages are non-monitored. This type of WF could be used in a system like XKEYSCORE [9], for example, where features of user traffic are scored to decide which ones to flag for storage or further analysis. The open-world model is in contrast to the closed-world model, where the client is not allowed to visit non-monitored pages. As there are almost a billion indexed web pages, the open-world model is of practical interest. The oldest WF attacks [5, 12, 15, 22] performed closed-world experiments on simple encrypted channels, and the newest [20, 24] give open-world results on Tor.

2.1 Closed world on encrypted channels

Cheng et al. [5] (1998), Sun et al. [22] (2002), and Hintz [12] (2003) published some of the earliest WF attacks on clients using a simple encrypted channel. In their works, the attacker was able to determine which TCP connection each packet came from, and at the time (before persistent HTTP connections) each resource on a web page was loaded on a different connection. Therefore, the attacker was aware of the byte length of each resource on the page. This is no longer a realistic assumption of the attacker, so further works (including ours) have weaker assumptions of the attacker's capabilities.

Liberatore and Levine [15] (2006), Bissias et al. [1] (2006) and Herrmann et al. [11] (2008) demonstrated successful WF attacks for such a weaker adversary. For their attacks, the attacker only needs to know the length of each packet. The attacker is not aware of which connection each packet belongs to. This is especially relevant to privacy technologies such as VPNs or TLS-enabled proxies. These works showed that, in the closed-world scenario, an attacker can distinguish among a few hundred web pages.

2.2 From encrypted channels to Tor

The above works showed that WF can succeed in the closed-world setting on a simple encrypted channel, such as TLS to a proxy, or a VPN. As suggested by Dyer et al. [8], such a channel can also be defended by using dummy packets to flood a constant stream of data in both directions (in an amount independent of the actual traffic), which is provably secure [24] against WF. However, both attacks and defenses suffer on low-latency, resource-limited anonymity networks, such as Tor. Expensive defenses are impractical as Tor is band-

Unauthenticated
Download Date | 10/23/16 6:26 PM

width starved [23], and attacks are harder [11] due to unpredictable network conditions and Tor's own defenses [24].

Panchenko et al. [20] (2009) demonstrated the first effective WF attack against Tor and other anonymity networks, using an SVM with a list of website features. They also showed a 57% TPR and 0.2% FPR in the open-world setting if the attacker monitors five pages. Dyer et al. [8] later showed that the variable n-gram classifier is also effective for WF with a similar set of features. Cai et al. [3] improved the closed-world accuracy by using the Damerau-Levenshtein distance. Their paper however did not perform open-world experiments. Wang and Goldberg [25] modified this algorithm and showed that the more accurate, modified version has an open-world TPR of 97% and FPR of 0.2%, but only for a single monitored page.

2.3 Further improvements to classification

The number of monitored pages used in early works on open-world WF [3, 20, 25] has been too small for implications on real-world practicality. Wang et al. showed [24] that their earlier attack [25] would have an open-world TPR of 83% and FPR of 6% when the number of monitored pages increased from 1 to 100. Cai et al. [2] showed a way to convert closed-world results to open-world results, which would result in a FPR of around 15% for the best attacks. Both of those are too high considering the low base incidence rate of the WF scenario.

Wang et al. [24] (2014) showed a new attack using the k-Nearest Neighbours (kNN) classifier and a new distance learning algorithm that achieved a significantly lower open-world FPR than previous work. For 100 pages, TPR was around 85% and FPR was around 0.6%. Furthermore, this attack is fast, as it takes only minutes to train and test 4,000 instances compared to several hundred CPU hours in some previous work [3, 25]. Newer advancemenets in page identification have further improved classification accuracy [10, 19]. Previous work has therefore shown that WF is effective in the open world under laboratory conditions; in this paper, we enhance such works with tools to operate in realistic settings.

3 Background

3.1 Tor and Tor Browser

Tor is a popular low-latency anonymity network supported by volunteer nodes that relay traffic for Tor users. Tor users construct a circuit of three nodes (the entry guard, the middle node, and the exit node) and use multi-layered encryption; the exit node is able to see the original TCP data sent by the client, and the entry node knows the client's identity. Without collusion between entry and exit nodes of the same circuit, none of the relays (or observers on those relays' networks) should be able to link the client's identity with their packets.

Tor sends data in fixed-size (512-byte) cells. Each circuit carries multiple streams corresponding to different TCP connections. For flow control, Tor uses `SENDME` cells, which are sent once per 50 incoming cells for each stream and per 100 incoming cells for each circuit.

Tor performs no mixing to keep latency minimal, which renders it susceptible to timing attacks. While it is well known that Tor's anonymity guarantees can be broken using a timing attack by global adversaries observing both ends of the user's circuit, it is assumed that such adversaries are rare due to Tor's global nature [7]. However, the WF scenario assumes a much weaker adversary: a local, passive adversary that only observes the user's end of the connection, such as the user's ISP, a packet sniffing eavesdropper, wiretapper, legally coercive forces, or any Tor entry node itself.

3.2 Website Fingerprinting

In website fingerprinting, a local, passive eavesdropper (such as an ISP) observes packets to and from a web-browsing client, and attempts to guess which pages the client has visited. For Tor, the eavesdropper is limited to only the time and direction of each Tor cell; as each Tor cell is encrypted and has a fixed size of 512 bytes, the attacker gains no further information from each cell.

In the open-world setting, the attacker has a set of pages that he is interested in (monitored pages), and he does not attempt to identify any other web page (non-monitored pages). The classification is positive when the attacker identifies any page as a monitored page (and negative for non-monitored); it is true if the identification is correct and false if not. Incorrectly classifying any monitored page as a different monitored page counts as a false positive. The open-world setting allows us to analyze a realistic attacker in a world where there may be any number of possible web pages. As the base incidence rate (the rate at which the client visits a page from the set of monitored pages) is expected to be low, the false positive rate must be low as well to avoid the base rate fallacy.

In this paper, we perform all experiments in the open world, and we obtain our results using leave-one-out cross validation.

Unauthenticated
Download Date | 10/23/16 6:26 PM

3.3 Experimental setup and evaluation

We used Tor Browser 3.6.4 with Tor 0.2.5.7 to collect the data used in this paper. We used a custom Firefox profile to enable automatic page loading and data collection, as otherwise Tor Browser launches its own instance of Tor and often interrupts page loading with error messages.

We collected data between September and October 2014. As several of our experiments required loading two pages at once in the same browser session, `tcpdump` did not provide us with enough information to distinguish between the two pages. We collected direct cell logs by modifying Tor to record stream and circuit numbers, and we use the direct cell logs only to obtain the ground truth for splitting experiments. Specifically, it should be noted that we only use those cell logs to obtain ground truth; this does not change the fact that any local passive attacker can perform splitting using our methods.

We disabled long-term entry guards to obtain a more complete view of the network. We used one client located at a fixed IP to collect our data. We collected several new data sets of raw packet traces:

1. Sensitive data set. Contains 40 instances of 100 sensitive pages that are censored in individual ISPs of the United Kingdom, Saudi Arabia, or China. This data set was also used by Wang et al. [24]; we updated the page list and recollected the data set because many of the pages are no longer accessible.
2. Open world data set. Contains one instance each of Alexa's top 5,000 pages. These pages are used as the non-monitored page set to test the false positive rate.

In the above data sets, we correctly split instances into separate pages upon collection. We also collected additional data to test splitting (Section 5) and background noise (Section 7), which will be described in detail at the start of the relevant sections.

4 Training Set Maintenance

In this section we demonstrate that it is practically feasible for a low-resource attacker to maintain an updated training set for WF attacks on Tor. Our experimental results show that:

1. Training set size: We empirically determine that a WF attacker can perform WF with a small training set. To do so, we examine how accuracy (TPR and FPR) varies with the size of the training set.
2. Training set update: We test several algorithms to update the training set, and show that a WF attacker can maintain the above small training set by updating all data points on a single desktop-class computer. The resulting training set is fresh enough that the attacker will lose very little accuracy due to the age of the training set.

Table 1. TPR and FPR for varying n_{inst} (between 0 and 90) and n_{nmsite} (between 0 and 9000). Each block contains the mean TPR (upper) and FPR (lower) for data points where n_{inst} and n_{nmsite} average to the respective row and column headings, and no data point contributes to two different cells. For example, the top-left block has data for $0 < n_{inst} \leq 10$ and $0 < n_{nmsite} \leq 1500$.

		n_{nmsite}					
		750	**2250**	**3750**	**5250**	**6750**	**8250**
	5	**0.235**	**0.163**	**0.229**	**0.229**	**0.253**	**0.244**
		0.001	**0.001**	**0.001**	**0.000**	**0.000**	**0.000**
	15	**0.641**	**0.646**	**0.653**	**0.621**	**0.621**	**0.627**
		0.007	**0.005**	**0.003**	**0.002**	**0.001**	**0.001**
	25	**0.722**	**0.709**	**0.703**	**0.699**	**0.696**	**0.693**
		0.011	**0.006**	**0.004**	**0.003**	**0.002**	**0.001**
	35	**0.753**	**0.753**	**0.747**	**0.743**	**0.737**	**0.733**
		0.011	**0.006**	**0.003**	**0.002**	**0.002**	**0.002**
n_{inst}	**45**	**0.776**	**0.767**	**0.765**	**0.756**	**0.752**	**0.751**
		0.017	**0.008**	**0.006**	**0.004**	**0.003**	**0.002**
	55	**0.792**	**0.785**	**0.781**	**0.777**	**0.773**	**0.771**
		0.019	**0.010**	**0.007**	**0.004**	**0.003**	**0.003**
	65	**0.801**	**0.794**	**0.788**	**0.786**	**0.781**	**0.779**
		0.027	**0.012**	**0.007**	**0.005**	**0.003**	**0.003**
	75	**0.811**	**0.804**	**0.801**	**0.796**	**0.794**	**0.791**
		0.019	**0.011**	**0.008**	**0.005**	**0.003**	**0.003**
	85	**0.814**	**0.809**	**0.806**	**0.804**	**0.801**	**0.798**
		0.018	**0.011**	**0.007**	**0.005**	**0.004**	**0.003**

4.1 Training set size

The number of packet traces the attacker needs to gather for the training set is $n_{msite} \cdot n_{inst} + n_{nmsite}$. Here, n_{msite} is the number of monitored sites (which we set to 100), n_{inst} is the number of instances of each site, and n_{nmsite} is the number of non-monitored sites used to train the classifier. (The client is of course allowed in the open world to visit other non-monitored sites, and indeed in our experiments will never visit a non-monitored site that was used for training.) The attacker controls n_{inst} and n_{nmsite} to improve the accuracy of classification. We are interested in knowing how large those variables must be to ensure accurate classification.

We performed experiments on Wang et al.'s kNN (with the number of neighbours set to 5) and the data set they used for their experiments as a basis of comparison with their experimental results [24]. Their data set contains 18000 elements (90 instances each of 100 sites, and 9000 non-monitored instances), collected throughout two weeks. We evaluate the ef-

Unauthenticated
Download Date | 10/23/16 6:26 PM

fect of both n_{inst} and n_{nmsite} on TPR and FPR, and we show the results in Table 1.

We can see that TPR and FPR increase with increasing n_{inst} (going down in each column) and decreasing n_{nmsite} (going left in each row). Furthermore, we find that a larger data set in general benefits TPR and FPR, but the benefit decays quickly with larger data sizes. Peculiarly, with the first two columns, FPR decreases after a point with increasing n_{inst}, possibly because greater information allowed the attacker to classify more accurately, overcoming the effect that increasing n_{inst} would bias the attacker towards making positive guesses. For instance, increasing n_{inst} from 35 to 85 and n_{nmsite} from 3750 to 8250 more than doubles the training set size, but only increases the TPR from 0.747 to 0.798 with no change to the FPR at 0.003. Having a smaller but more frequently updated training set can be beneficial to the attacker. Previously, Wang et al. [24] had only studied the effect of varying n_{nmsite} on TPR and FPR.

The optimal data set size to use depends on the attacker's specific need. We present the accuracy at a specific point, where $n_{nmsite} = 100$, $n_{inst} = 31$ and $n_{nmsite} = 3700$, where the accuracy is high enough to be a threat to web-browsing privacy.[2] At this point the TPR is 0.77 ± 0.01 and the FPR is 0.003 ± 0.001. This requires a total of only 6800 elements in the data set, instead of all 18000 elements. If the attacker loads more web pages, the accuracy still increases, but at a slowing rate. The attacker can further trade off an increased TPR for an increased FPR by intentionally including fewer non-monitored data points (so neighbours of each point are less likely to originate from the non-monitored class) or by decreasing the number of neighbours (as a page can only be classified as a monitored page only if all of its neighbours originated from that page). Therefore, we have shown that the attacker can maintain high accuracy with a fairly small data set.

4.2 Training set update

In the above, we showed that an attacker can classify 100 sites accurately by maintaining a data set with approximately 6800 elements in it. We are interested in knowing whether or not maintaining a fresh data set of such a size is practical.

While collecting this data set, the mean amount of time to load each page was 12 s, which would mean that the whole training set could be re-collected once per 0.9 days on a single dedicated machine. We have experimentally confirmed that this can be done. If we were to continuously load data and update the data set with a single dedicated machine, the data set itself will be on average half a day old for classification.

It is interesting to study how the accuracy of the kNN attack would change based on the age of the data set. In previous work, Juarez et al. observed a decrease in accuracy (in a different algorithm) if data was collected 10 days apart, but they did not experiment on fresher data [13]. We will experiment on fresher data to observe a finer change in accuracy. The data set of 18000 elements was collected over a span of two weeks, and we will vary the chosen training set in order to test the effect of age on accuracy. We separate out the 2000 newest elements as the testing set, leaving 16000 potential elements for the attacker for his training set. Note that the results in this section are therefore necessarily worse than the results above: none of the 2000 newest elements are available to the attacker at all, creating a minimal time gap of around a day.

The attacker follows a simple algorithm in order to maintain a training set of $n < 16000$ elements:

1. The attacker starts with the oldest n elements as his training set.
2. Each other element in the rest of the set is presented to the attacker, in order of age (oldest first). For each element, the attacker decides whether or not to update, by including the element in his training set.
3. If the attacker decides to update, the attacker takes the element, reads it, and throws away one element from his training set, maintaining its size.

Note that in Step 2 above the attacker is not allowed to look at the element before deciding whether or not to take it, since looking at the element means loading the page. The attacker's options in updating the training set are therefore limited. We consider three simple training set update schemes:

1. No update: The attacker never updates his training set, keeping the oldest.
2. Update oldest: For each element, the attacker takes the element with probability p, and then throws away the oldest element in his training set. For $p = 1$, this is equivalent to choosing the newest elements of the training set.
3. Update least consistent: For each element, the attacker takes the element with probability p, and then throws away the least consistent element in his training set. The attacker calculates the consistency score of each element as follows: the consistency score of an element is the number of neighbours that belong to the same class, out of its 100 closest neighbours. The higher the number, the more consistent that element is.

When the attacker chooses the newest elements of the training set (which are approximately one to three days old), the TPR

[2] Our rationale for choosing this specific point is that it represents a reasonable data set size, and gives a sufficiently threatening accuracy value.

Unauthenticated
Download Date | 10/23/16 6:26 PM

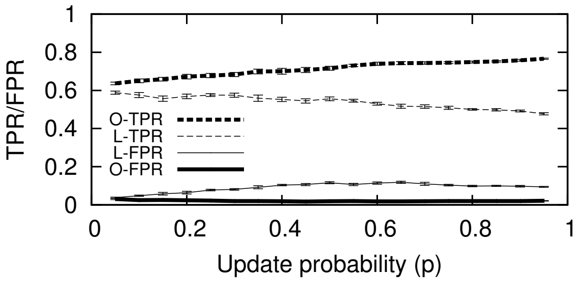

Fig. 1. TPR and FPR when p varies between 0 and 1 for two training set update schemes: updating the oldest elements first ("O-TPR" and "O-FPR"), and updating the least consistent elements first ("L-TPR" and "L-FPR").

is 0.77 ± 0.01 and the FPR is 0.023 ± 0.002. We note that the FPR has increased significantly compared to the result in the previous subsection, despite having a training set of the same size: the difference is that in our experimental setup; here, the attacker is not allowed to include the latest 2000 elements at all in his training set, whereas in the previous subsection the attacker could include all elements but the testing element. With no update for two weeks, the TPR is 0.62 ± 0.01 and the FPR is 0.035 ± 0.003, which is markedly weaker though still threatening to privacy.

In Figure 1 we show the accuracy (both TPR and FPR) of each scheme when varying p, along with the error bars resulting from the randomness of selecting elements with probability p. There is a steady increase in accuracy with increasing probability p if the attacker should update the oldest elements first; the correct p to choose depends on the attacker's resources, though as earlier noted, even $p = 1$ is feasible for a single dedicated machine to maintain a training set size of 6800; a lower p proportionally reduces the amount of time the attacker must spend in maintaining the training set, or it opens up the option of maintaining a larger training set. Interestingly, updating the least consistent training points first actually decreases the accuracy of the scheme, with a notably high false positive rate, and the accuracy worsens when more updating is done. We hypothesize that throwing out inconsistent points makes the training set "closed-minded": all retained points are close neighbours of each other, limiting the perspective offered by a fuller and fresher data set. We tested several other schemes for attempting to throw out "bad" training points first, and all such results proved similarly less effective than simply throwing out the oldest elements first.

Juarez et al. showed that if the classifier trains on one version of Tor Browser but tests on another, the accuracy may deteriorate. [13] While Tor Browser displays warnings to remind its clients to update their browsers, some clients may nevertheless continue to use outdated versions. In the worst case, the attacker can collect data corresponding to the client's browser version. This increases the size of the data set the attacker must maintain, which, for the same attacker resources, makes the data set less fresh, leading to a loss in classification accuracy. An outdated client is more difficult to attack with website fingerprinting.

5 Splitting Algorithms

Current WF techniques only accept cell sequences corresponding to a single page as input; under laboratory conditions, the researcher uses the ground truth of the data collection system to decide where to split the full sequence of cells into single-page cell sequences. These attacks cannot operate in the wild unless we can split accurately without this ground truth. To verify this claim, we combined pairs of random web pages in Alexa's top 100 sites, and asked the kNN classifier to identify one of them. In this experiment, each class corresponds to one real web page, and each instance of the class is a packet sequence including the real page and another randomly chosen page. The kNN classifier had a drastically lowered TPR of 15% when attempting to classify this site, indicating that splitting is necessary for us to recover the accuracy of the kNN classifier.

In this section, we tackle the splitting problem. Solving the splitting problem incorrectly could result in a cell sequence with extra or fewer cells (harder to classify); missing a split altogether almost certainly results in two negatives (two pages classified as a non-monitored page that the attacker is not aware of). We explain the terminology used in this paper, outline our strategy to solve the splitting problem, and then discuss the specific algorithms we use. In Section 6 we will see the results of the algorithms.

5.1 Terminology

In this work we introduce some new terminology in order to explain our splitting solution.

Tor delivers data in fixed-size Tor cells on the application layer before it is packaged by TCP, and so our splitting algorithm uses sequences of Tor cells rather than raw TCP/IP packets. To do so, we reconstruct TLS records from TCP segments and infer the number of Tor cells from the record length of each TLS record. When a user visits a web page, the sequence of incoming and outgoing Tor cells that result from the page visit is referred to as a *cell sequence*. A single cell sequence corresponds to a single web page. The user may visit many web pages over a long period of time (for example, an

Unauthenticated
Download Date | 10/23/16 6:26 PM

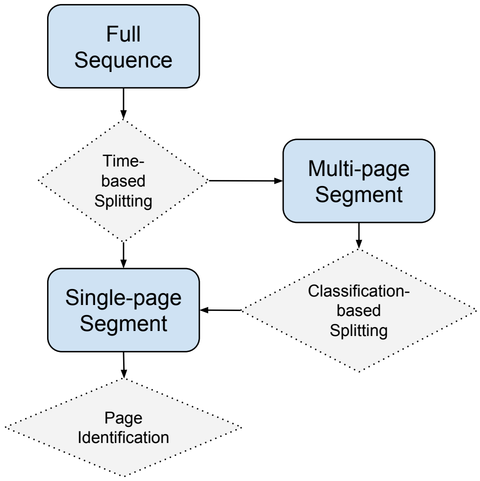

Fig. 2. Splitting process, with ideal results. Overall, the attacker wants to obtain single-page segments from the full (raw) sequence.

hour), and the attacker collects the sequence of incoming and outgoing cells as a *full sequence*.

In splitting, the attacker wants to divide the full sequence into *cell segments*. The ultimate goal is to have cell segments that each contain a single cell sequence (corresponding to one web page). These are referred to as *single-page segments*. Cell segments that contain more than one page, possibly in error or as a temporary transitional state between the full sequence and single-page segments, are referred to as *multi-page segments*.

To obtain single-page segments, the attacker needs to find the correct splits. A split is a location in the full sequence where the cells before the split and the cells after the split belong to different web pages. In multi-page segments where multiple pages may overlap, we define a split as the location of the first cell of the latter page. In such a case, splitting correctly would still result in some overflow of cells into the latter segment. We do not attempt to split between different streams (for example, different images or scripts) of the same web page.

5.2 Splitting process

There are three steps in our splitting process, as shown in Figure 2:

1. **Time-based splitting**. The full sequence is split with a simple rule: if there is a time gap between two adjacent cells greater than some amount of time t_{split}, then the sequence is split there into individual cell segments.
2. **Classification-based splitting**. Time-based splitting may not be sufficient to split multi-page segments with a smaller time gap than t_{split}. We use machine learning techniques both to decide whether or not to split them further (*split decision*) and where (*split finding*).
3. **Page identification**. The objective of the first two steps is to split the full sequence into single-page segments as accurately as possible. Then, we may attempt to identify the page corresponding to the cell segment. In this work we will investigate the effect of background noise, incorrect splitting, and incomplete pages on known page identification techniques, but developing more powerful page identification techniques is outside the scope of this work.

The following types of segments may result from time-based splitting:

1. Single-page segments. In this case we should not attempt to split the segment further.
2. Two-page segments. We attempt to split these cell segments by finding the point where the second page starts.
3. Multi-page segments containing three or more pages.

We apply classification-based splitting in two steps in order to split two-page segments further into single-page segments. First, in *split decision*, we attempt to distinguish between single-page and two-page segments with machine learning. Then, in *split finding*, we take two-page segments and find the optimal location to split them. To find a split in a two-page segment, the classifier looks at each outgoing cell and assigns a score based on features of its neighbouring cells. The classifier guesses that the maximally scored cell is the correct split.

We do not split multi-page segments containing three or more pages. Our methods may apply to these segments as well, but the accuracy would be lower, and it complicates our presentation. Rather, in Section 6.2 we show how we minimize their occurrence probability as an explicit strategy.

5.3 Time-based splitting

Cell sequences from page loading may be separated by a time gap during which there is no web activity. We therefore split the full sequence at all points in the sequence where no traffic is observed for some amount of time t_{split}. Our choice of t_{split} seeks to minimize the chance of splitting single-page segments, which should not be split any further. A larger t_{split} reduces such a risk but renders multi-page segments more likely if the client's *dwell time* is small.[3] To split these we need to apply classification-based splitting, which we discuss in the next section.

To obtain the correct t_{split} we consider two potential errors resulting from splitting with t_{split}:

[3] The dwell time is the amount of time a user stays on a page between two page visits.

Unauthenticated
Download Date | 10/23/16 6:26 PM

1. Splitting a single-page segment with t_{split}. The attacker should not split single-page segments further. We will consider the consequences of such a split: it is still possible to classify a cell sequence correctly even with only part of the cell sequence.
2. Failing to split a multi-page segment with t_{split}. The probability of this error occurring depends on dwell time. We must proceed to classification-based splitting in order to split two-page segments, which is less accurate than a simple time-based rule.

We want either source of error to be unlikely. A smaller t_{split} increases the chance of the former and decreases the chance of the latter, and vice-versa, so a suitable value must be chosen. We will show how we choose t_{split} and how it affects accuracy values in Section 6.2.

5.4 Classification-based splitting

Cell segments used in classification-based splitting can consist of web pages organized in four possible ways:

Class 1. Two pages, positive-time separated. This is where the user dwells on a web page for an amount of time before accessing the next, thus causing a lull in web activity and a noticeable time gap in traffic. This noticeable time gap is, however, less than t_{split} used by time-based splitting; otherwise it would have been split in time-based splitting.

Class 2. Two pages, zero-time separated. This is where the user clicks on a link from a web page that is loading, thus halting the web page and sending out requests for the next immediately, so that there is a clear division between two web pages but it is not marked by a time gap.

Class 3. Two pages, negative-time separated. This is where the user is loading two pages at once in multiple tabs. In this case, we consider a correct split to be the time at which the second page starts loading. This is the hardest class to split as there is no noticeable gap nor a clear pattern of cells indicating the gap. However, we can still split cell sequences in this class using machine classification by extracting useful features. We list our feature set in Appendix A.

Class 4. One page. In this case time-based splitting was sufficient to isolate a page into its own cell segment. We need to avoid splitting such a page.

To split two-page segments properly is a two-step process. The first step is *split decision*, where we distinguish between two-page segments and single-page segments. This is necessary to perform the second step, *split finding*, where we find the split in two-page segments.

Split decision. The algorithm for split decision takes as input a page segment, and returns a binary "yes" (it is a two-page segment) or "no" (it is a single-page segment). It is trained on two classes: a class of two-page segments, and a class of single-page segments. If split decision returns "no", we believe the sequence comes from class 4, so we skip split finding and go straight to page identification. For *split decision*, we tried `kNN`, `Time-kNN`, and `SVM`:

1. `kNN`: kNN with features and weight learning. This is similar to the approach used by Wang et al. for website fingerprinting [24], where features are extracted from the cell segment and used by a weight learning algorithm that determines the distance function, which is used by a kNN classifier. We tested this algorithm with their original feature set.
2. `Time-kNN`: Time-based kNN. We added a number of features to the above that are related to inter-cell timing. These include the largest and smallest inter-cell times in the cell segment, the mean and standard deviation for inter-cell timing, and others.
3. `SVM`: SVM with features. This is similar to the approach used by Panchenko et al. for website fingerprinting [20]. The chief difference is that we choose a different cost and gamma value (as this is a different problem); in addition, we do not append the entire cell segment onto the feature list. We selected parameters for the SVM as it is highly sensitive to incorrect parameters. We chose the radial basis function with $\gamma = 10^{-13}$ and cost for incorrect classification $C = 10^{13}$ to maximize accuracy.

Split finding. When the split decision algorithm returns "yes", we move on to *split finding*. The correct split location is the point at which the second page begins loading (i.e., an outgoing request cell is sent to the server of the second page). To find the correct split, we score every outgoing cell in the cell segment based on its neighbourhood of cells, and return the highest-scoring outgoing cell as the location of the guessed split. The split-finding algorithm takes as input a cell and its neighbourhood of cells, and returns a score representing its confidence that this cell marks the start of the second page. For *split finding*, we tried `kNN`, `LF-kNN` and `NB`:

1. `kNN`: A kNN classifier with a scoring system. As the features used by Wang et al. [24] are not suitable for splitting, we used a set of 23 features based on timing and cell ordering, given in Appendix A. We score each candidate cell by finding 15 closest neighbours: Neighbours from the "correct split" class increase the score and neighbours from the "incorrect split" class decrease the score.

Unauthenticated
Download Date | 10/23/16 6:26 PM

We guess that the highest-scoring candidate cell is the real split.

2. `LF-kNN`: A kNN classifier that uses the last cells before splits and first cells after splits to classify elements. The classifier recognizes four classes: *correct-before*, the last 25 cells before a correct split; *correct-after*, the next 25 cells after a correct split, and similarly *incorrect-before* and *incorrect-after* for incorrect splits. When evaluating a candidate split, the last 25 cells before the candidate split are scored against *correct-before* and *incorrect-before* and the next 25 cells are scored against *correct-after* and *incorrect-after*. All four scores are then added together for a final score.
3. `NB`: Naive Bayes with features. The Naive Bayes classifier involves explicit probabilistic scoring, which is suitable for this purpose, and it was used successfully by Liberatore and Levine [15] for WF. Features are trained and tested with the assumption that they follow independent normal distributions. We use the same feature set as `kNN` above. Each potential split will have a score indicating the possibility that it belongs to the first class, and the potential split with the highest score is picked out.

We based our methods on attacks that succeeded for WF as we find that there are similarities between splitting and WF. Specifically, both are classification problems that take web packet sequences as input, and relevant features include packet directions, timing, and ordering. We did not attempt to use SVMs with Damerau-Levenshtein distance (like Cai et al. [3]) because the Damerau-Levenshtein distance ignores timing, and timing is important in finding splits.

6 Splitting Results

In this section, we experimentally validate our splitting algorithms and show their accuracy. The main results are:

1. **Time-based splitting** (Section 6.2). Empirically, we find that we can perform time-based splitting with $t_{split} = 1$ s, at no cost to page identification accuracy. We will discuss previous research, which suggests that most web page accesses have a higher dwell time than this value.
2. **Classification-based splitting** (Section 6.3). If the above fails to split a two-page segment (i.e., the time gap is smaller or did not exist), we find that we can still perform *split decision* to identify two-page segments and *split finding* to split them correctly with high accuracy.

Combined, these results mean that we can split full sequences into single-page segments with high accuracy.

6.1 Experimental setup

We collected data as described in Section 3.3. To acquire ground truth for splitting, we instrumented Tor to log cell information: in particular, we needed the stream ID and data type of each cell.[4] To distinguish between cells from two different pages, we recorded the time when the request for the second page was sent, and marked the first outgoing `STREAM BEGIN` cell at or after that time as the start of the second page. All new streams started after that cell were marked as belonging to the second page, whereas streams before that cell were marked as belonging to the first page. This allows us to record ground truth of which page each cell belonged to.

We loaded zero-time separated pages (class 2) by loading two pages in the same tab, and we loaded negative-time separated pages (class 3) by loading two pages in different tabs, with a time gap between 5 and 10 seconds. We chose this time gap to give enough time for the first page to start loading, and to ensure that the chance that the first page finishes loading before the second page starts is small. If the first page finishes loading before the second page starts, then these two pages are actually separated by a positive time gap. We processed all cell sequences to find positive-time separated two-page segments this way, and moved them from their original classes to positive-time separated pages (class 1). Overall, the number of elements in classes 1, 2, 3 and 4 were about 1800, 1200, 1500, and 1600 respectively.

6.2 Time-based Splitting

The first step in processing the full sequence into single-page segments is to split the full sequence with a simple time-based rule: if the difference in time between two cells exceeds t_{split}, then we split the sequence there.

We argued in Section 5.3 that there are two opposing factors affecting the choice of t_{split}:

1. Increasing t_{split} decreases the chance of splitting single-page segments, which should not be split.
2. Decreasing t_{split} increases the chance of splitting multi-page segments, which should be split.

We will present our experimental results in this section accordingly. First, we will increase t_{split} until there is almost no chance of splitting single-page segments. Then, we will decrease t_{split} to increase the chance of splitting multi-page segments. It may seem that those two steps should be taken in

[4] Any local, passive attacker can use our splitting method from `tcpdump` info; we only need cell data for experimental ground truth.

Unauthenticated
Download Date | 10/23/16 6:26 PM

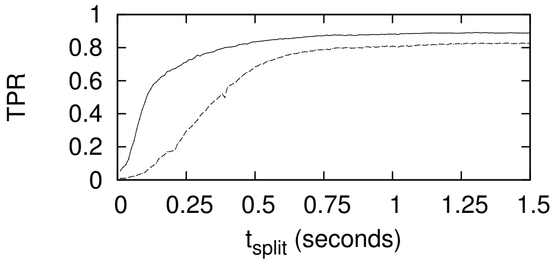

Fig. 3. TPR of kNN when splitting cell sequences with t_{split}. The solid line represents an attacker who adjusts for splitting by splitting his own training set, and the dashed line represents an attacker who does not adjust for splitting.

tandem with each other to find an optimal solution; however, we will in fact find that t_{split} is so small after the first step that we do not need to take the second step at all.

6.2.1 Increasing t_{split}

To experimentally determine the suitable t_{split}, we applied time-based splitting to single-page segments, and tested page identification with the kNN classifier by Wang et al. [24] on the resulting segments while varying t_{split} from 0.01 s to 1.5 s. We ran our experiments on the splitting data set, on which the TPR of kNN was 0.89.

If t_{split} splits a single-page segment into several segments, we chose the largest segment to keep to classify, and assign the smaller segments to the non-monitored class. The attacker applies splitting to his own training set as well, under the same t_{split}, in order to maximize his accuracy in classifying split segments; we will also consider an attacker who does not change his training set. We do not make any adjustments to kNN itself for splitting.

In Figure 3 we plot the resultant negative effects on the true positive rate (the effect on the false positive rate is very small). We see from Figure 3 that classification accuracy reaches the maximum after around $t_{split} = 1$ s. At this value, the TPR is 0.88 compared to the maximal 0.89. There are in fact time gaps still greater than 1 s, but removing them did not affect classification accuracy.

If the attacker does not adjust, the accuracy is significantly lower. For example, at $t_{split} = 0.5$ s, splitting with adjustment gives a TPR of 0.83, while splitting without adjustment gives a TPR of 0.68. At $t_{split} = 1$ s they were 0.88 and 0.81 respectively. It is realistic to allow the attacker to apply splitting to his data set.

We have therefore found that there is almost no loss in classification accuracy with $t_{split} = 1$ s. The implication is that even if the client only waits 1 s between two page loads, we can distinguish between them by time-based splitting. Next, we want to know if we should reduce t_{split} any further.

6.2.2 Decreasing t_{split}

Figure 3 suggests that t'_{split} can be reduced below 1 s with only a small loss in TPR, so it may be argued we should reduce t_{split} below 1 s, to trade an increased chance of splitting single-page segments for an increased chance of splitting multi-page segments.

Reducing t_{split} to some $t'_{split} < 1$ s has a benefit if the client's dwell time t_{dwell} is between t'_{split} and t_{split}. Recall that classification-based splitting does not handle segments with three or more pages, which will occur if there are two or more dwell times less than t_{split} in a row; we want this situation to be rare after time-based splitting, so we want to know the probability $t'_{split} < t_{dwell} < 1$ s.

As we cannot collect information on Tor clients, we do not know the true dwell time distribution of Tor clients, so we must defer to previous work on dwell time for web traffic (without Tor). It has long been established that dwell time has a heavy tail [6]. Previous authors have found dwell time to be well-fitted by Pareto [4], lognormal [18] and "negative-aging" Weibull [16, 18] distributions with the mean being around 60 s. The probability of 0 s $< t_{dwell} < 1$ s is thus very small, so any choice of t'_{split} is not likely to produce a benefit. This also implies that the probability of failing to split a time gap twice in a row (for multi-page segments with three or more pages) will be even smaller.

Furthermore, we will next see that when there exists a small time gap between packet sequences (under 1 s), denoted as Class 1 in our terminology (see Section 5.4), classification-based splitting is very accurate. This further contributes to our argument that reducing t_{split} further would produce no notable benefit in splitting multi-page segments.

6.3 Classification-based splitting

In this section, we present our results on splitting two-page segments with classification. We used two metrics to evaluate the effectiveness of splitting:

1. Split accuracy. We consider the split correct if it is within 25 incoming and outgoing cells of the correct split. We choose 25 as a range range within which the page identification TPR remains above 50–60% (we will show this later). Randomly guessing any outgoing cell as the correct split will result in a split accuracy of 4.9% (computed from our data set). We also show the standard deviation of

Unauthenticated
Download Date | 10/23/16 6:26 PM

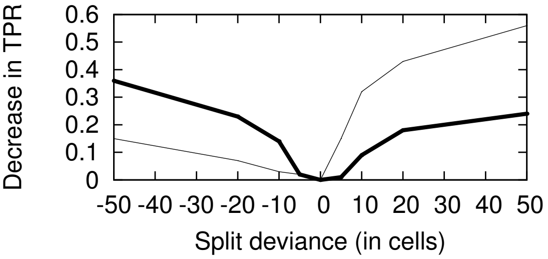

Fig. 4. Absolute decrease in TPR if the split is guessed incorrectly. A negative split deviance indicates that the guessed split was before the correct split, and a positive split deviance indicates it was after. There are two lines: The thicker line is for the segment that received extra cells because the split was in the wrong place; the thinner line is for the segment that lost cells. The false positive rate did not change significantly.

Table 2. Split decision accuracy. The number in row i, column j is the probability that the classifier thought an element of class i belonged to class j. A greater value in the diagonal is better.

		Class 1/2/3	Class 4
kNN	Class 1/2/3	0.92 ± 0.04	0.08 ± 0.04
	Class 4	0.022 ± 0.006	0.978 ± 0.006
Time-kNN	Class 1/2/3	0.96 ± 0.05	0.04 ± 0.05
	Class 4	0.03 ± 0.02	0.97 ± 0.02
SVM	Class 1/2/3	0.93 ± 0.01	0.07 ± 0.01
	Class 4	0.10 ± 0.03	0.90 ± 0.03

split accuracy by choosing random subsets of the data set for training and testing.

2. Split deviance. This is the number of cells the guessed split was from the correct split. We present the three quartiles of this value. A larger deviance increases the difficulty of page identification.

We measured the effect of split deviance on page identification accuracy with Wang et al.'s kNN. First, we took the last ℓ cells from a single-page segment and prepended it to another single-page segment, and tested the effect of varying ℓ on the accuracy of identifying either page; this simulates the situation where the split was too early. Similarly we tested the effect of varying ℓ on accuracy when the split was too late. We show the results in Figure 4. We see that under 5 cells the absolute decrease in TPR is negligible, with the exception that a cell segment which lost its first few cells is significantly harder to classify (due to the fact that the kNN uses the first few cells as an important feature). Under 25 cells the decrease in TPR is around 20% except for this specific case.

Table 3. Split finding accuracy. We show the accuracy for each of the three types of two-page segments (Class 1: positive-time, Class 2: zero-time, Class 3: negative-time). We show the first, second, and third quartiles of the absolute split deviance in parentheses, separated with slashes.

	Class 1	Class 2	Class 3
kNN	0.92 ± 0.02 $(0/0/1)$	0.66 ± 0.04 $(2/8/59)$	0.34 ± 0.04 $(9/87/332)$
LF-kNN	0.94 ± 0.01 $(0/0/2)$	0.61 ± 0.03 $(3/13/53)$	0.18 ± 0.02 $(60/205/526)$
NB	0.16 ± 0.03 $(67/339/1507)$	0.09 ± 0.02 $(237/851/2126)$	0.04 ± 0.02 $(388/1385/3922)$

6.3.1 Split decision

We evaluate the accuracy of *split decision*: distinguishing between single-page segments and two-page segments. We collected 5,000 two-page segments by randomly choosing pages from the sensitive site list and accessing them one after the other. We also chose 5,000 single-page segments from the sensitive site list. We tested kNN, Time-kNN, and SVM (described in Section 5.4) for this problem, and show the split accuracy and absolute split deviance.

The three classifiers had similar performance, except that SVM had trouble identifying class 4 (single-page segments). Since identifying class 4 as class 1/2/3 means that it will undergo splitting by the split finding process, and splitting a single-page segment makes classification significantly harder, we want class 4 to be identified correctly. Therefore the other two methods are superior; we chose Time-kNN for further experiments. We present the complete data in Table 2.

6.3.2 Split finding

We evaluate the accuracy of *split finding* in two-page segments. We tested kNN, LF-kNN, and NB for this problem (described in Section 5.4), and similarly present the split accuracy and absolute split deviance.

For both cases, our results showed that NB was only slightly better than random guessing, even though it used the same features as kNN. This is possibly because kNN was more tolerant to bad features than NB, as the kNN weight-learning process filtered out bad features. LF-kNN was overall slightly worse than kNN in terms of finding the correct split. This may be because LF-kNN is only allowed to train on a maximum of 25 cells before and after true splits, so it has no access to features relating to the remainder of the cell sequence (such as total number of cells in the sequence). We present the complete data in Table 3.

Unauthenticated
Download Date | 10/23/16 6:26 PM

Table 4. Overall split accuracy for the best algorithms (`Time-kNN` for split decision, `kNN` for split finding). Classes 1, 2, and 3 are positive-time, zero-time, and negative-time separated two-page segments respectively; Class 4 is single-page segments. For class 4, split accuracy is simply the chance that it was identified as class 4 (i.e. split decision returned "no").

Class 1	Class 2	Class 3	Class 4
0.88 ± 0.05	0.63 ± 0.05	0.32 ± 0.04	0.97 ± 0.02

6.3.3 Together

Finally, we combine all the results in this section and present split accuracy using the best algorithms (`Time-kNN` for segment classification and `kNN` for split finding) in Table 4. The table shows that classification of Class 3 is indeed the most difficult: in fact, we cannot expect to correctly split overlapping pages.

7 Removing Noise

In this section, we will discuss another aspect of realistic cell sequences that may hamper website fingerprinting: background noise. The client may, for example, download a file, listen to music, or watch a video while browsing. These activities are persistent and may interfere with both splitting and page identification if the noise bandwidth rate is high enough. We characterize noise in Section 7.1, and we demonstrate that only noise requiring high bandwidth ("loud" noise, such as videos and streams) can affect website fingerprinting. Then, we present two strategies for the website fingerprinting attacker to adjust for loud noise:

1. Removing noise from testing elements (Section 7.2). We propose a number of classification-based and counting-based approaches to remove loud noise. We find that many of them fail to remove noise accurately. Analytically, we show that there are inherent difficulties in removing loud noise due to variation in intercell timing.
2. Adding noise to training elements (Section 7.3). Instead of removing noise from testing elements, the attacker can attempt to add noise to training elements for more accurate classification. The accuracy of such schemes is dependent on the ability of the attacker to generate or find the type of noise used by the client, and we present our results in terms of the attacker's ability to do so.

We tested our algorithms against two types of loud noise: audio streaming and file downloading noise, and we will show that we can still successfully classify under such noise.

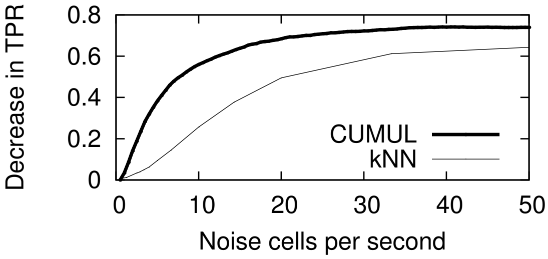

Fig. 5. Absolute decrease in TPR of the CUMUL and kNN classifiers when random noise is added.

7.1 Characterizing loud noise

We first characterize loud noise by identifying the number of noise cells per second that is necessary for WF accuracy to drop. To do so, we add random cells to cell sequences. Then, we ask the CUMUL classifier [19] and kNN classifier [24] to identify the original page. The attacker in this case is not aware of the noise and does not attempt to adjust for it; his training set has no noise cells. This is a conservative assumption for the attacker's accuracy. We add noise cells at b_{noise} cells per second, with each intercell time selected uniformly randomly between 0 and $2/b_{noise}$ seconds.

We show the results in Figure 5. CUMUL is more sensitive to noise, and its accuracy decreases more quickly. For kNN, the absolute decrease in TPR becomes significant after 5 cells per second (20 kbps), and the TPR has dropped significantly at around 20 cells per second (80 kbps). Video streaming, audio streaming and file downloading may exceed this bit rate, while other sources of noise such as chatting, AJAX and Tor protocol overhead may not. For the rest of the section, we will examine two types of noise: audio streaming noise, and file downloading noise.

7.2 Removing noise from testing elements

We collected two instances of noise over Tor. First, we downloaded a 10 GB file from a web site that offered speed testing for users (file download noise). Second, we downloaded 10 minutes of audio from a music site (audio noise). Considering the number of cells loaded over time for both types of noise, our audio noise is a step function, whereas our file download noise is continuous.

We use the sensitive data set to obtain packet sequences from web browsing, and we refer to cells from this data set as real cells. We merged the cell sequences for noise cells and real cells (keeping timing and ordering), and attempted to remove only the noise cells from the merged cell sequence. We attempted two approaches to noise removal: the classification-

Unauthenticated
Download Date | 10/23/16 6:26 PM

based approach, and the counting-based approach. In the classification-based approach, the attacker attempts to use machine learning to decide whether a given cell is noise or not. In the counting-based approach, the attacker counts the total number of cells per interval of time and removes a number of noise cells in each interval.

7.2.1 Classification-based approach

First, we show a classifier that failed to remove noise accurately, using feature extraction in an approach similar to splitting and WF. For this experiment, we scaled down the file download rate to the web browsing rate by multiplicatively increasing the intercell time, so that the number of cells of each class is about equal: otherwise, the attacker could obtain high accuracy in removing file download noise just by removing all cells.

This problem is similar to split finding, so we will use a similar algorithm: we extract a neighbourhood of cells from each candidate cell, and use a scoring classifier to score the candidate cell. We used 65 features, similar to the features we used for split finding, and we give the full list in Appendix A. To attempt to classify this set, we used SVM with the radial basis kernel, and parameters $\gamma = 10^{-13}$ and cost for incorrect classification $C = 10^{13}$ to maximize accuracy. We chose SVM because it has been used successfully in the past for website fingerprinting [3, 20].

There were two classes: noise cells and real cells. With 400 testing and 400 training elements for each, we ran the SVM 100 times on random subsets of the sensitive data set (which has 4800 elements). The accuracy was 67 ± 10% for both types of noise. These values are low compared to what is necessary for accurate page identification after noise removal. Figure 5 suggests that at around 20 cells per second (about 8% of web-browsing traffic rate) page identification deteriorates significantly. If the noise cell rate is about equal to the real cell rate, we found that the classification accuracy needs to be above 92% for the kNN attack to succeed. Alternatively, the attacker needs to use a more noise-resilient classification algorithm. We have therefore shown that our attempt at a classification-based approach failed to remove noise.

7.2.2 Counting-based approach

We devised two counting-based algorithms, one for removing continuous noise and one for removing step-function noise. We reduced the file download noise rate to the same as the audio noise rate, so that a comparison could be made between continuous noise and step-function noise. Then, we mixed real cells and noise cells together, as above.

Table 5. Noise removal accuracy for our counting-based algorithm. Noise and real cells removed are percentages of their totals. A higher value for the former and a lower value for the latter indicates more accurate noise removal.

	Noise cells removed	Real cells removed
File download	$55\% \pm 16\%$	$39\% \pm 21\%$
Audio	$66\% \pm 14\%$	$32\% \pm 12\%$

For continuous noise, we calculated the noise cell rate, and attempted to remove the noise by removing 1 incoming cell every t seconds, where $1/t$ is the observed noise rate. More precisely, we divided the noisy cell sequence into portions of t seconds, and removed the first incoming cell from each portion. If there was no cell within a portion, we also attempted to remove an extra incoming cell from the next portion.

To remove step-function noise, we used an algorithm that learned the properties of each step, including the average duration of the step, the number of cells in the step, and the amount of time between steps. Then, we removed noise with a counting-based algorithm similar to the one for continuous noise, parametrized by the properties we learned.

We show the results in Table 5. The results show very poor accuracy in removing noise, even though we verified that we learned the parameters of the noise correctly: our counting-based algorithms removed significant numbers of real cells. This can be compared to a baseline of 50% for file download noise and 42% for audio noise if cells were simply removed randomly, at a rate proportional to the total number of noise cells and real cells. Results from both counting-based algorithms are unrecognizable by the kNN page identification classifier (accuracy is close to random guessing).

7.2.3 Difficulties in noise removal

We identified two reasons why both our classification-based algorithm and our counting-based algorithm failed.

1. **Short-term cell rate variation.** Specifically for file download noise, we found a very large variation in intercell time. The mean time was 0.0004 s, but the standard deviation was 0.003 s which is seven times higher. Classification-based algorithms may also be confused by such an inconsistent feature. If we were to use a counting-based algorithm that removes one cell per 0.0004 s in accordance with the mean noise rate, we would incorrectly

Unauthenticated
Download Date | 10/23/16 6:26 PM

Table 6. Accuracy when attacker adds noise to his training set, when classifying testing elements with noise.

Noise type	TPR	FPR
Audio	0.686 ± 0.007	0.026 ± 0.003
File download	0.481 ± 0.008	0.033 ± 0.003

remove a real cell or fail to remove a noise cell with high probability.

2. **Long-term cell rate variation.** We also found that there was significant variation in the number of cells sent over time in the long term. We took our noise data and calculated the number of cells sent for every 5 second interval. For file download noise, we observed a mean of 11800 ± 1200 cells per 5 seconds; for audio noise, we observed 540 ± 30 cells per 5 seconds. Both of these variations are high enough that the attacker may not be able to guess the cell rate correctly, thus failing to remove significant amounts of noise or removing significant numbers of real cells.

7.3 Adding noise to training elements

In the above, the attacker attempted to remove noise from packet sequences collected from the client. This did not work well and resulted in poor classification accuracy. In this section, we investigate a different strategy, where the attacker adds the client's background noise to his training set, and we evaluate the effectiveness of this approach.

The attacker does not need to identify the type or origin of the client's background noise. Instead, he simply takes background noise packets directly from the client's packet sequence, and adds them to his training set with the same ordering and timing. To match the length of packet sequences in his training set, he either cuts or expands the noise sequence in a trivial manner.

We experiment with an attacker who adds noise to his training set for both audio noise and file download noise. To measure classification accuracy, we use the sensitive data set. In Table 6, we show the accuracy of the classifier when the attacker adjusts his training set as above.

We see that an attacker can still identify pages somewhat accurately if he adds noise to his data set. The bandwidth rate of file download noise (9 Mbps) was much higher than that of audio noise (680 kbps). This suggests that the attacker can still classify packet sequences with high levels of noise by finding the noise packets from the client's packet sequence and adding them to his data set. Some attackers may find it difficult to do so; for example, an attacker who tries to block web page accesses as they happen or shortly after they happen may not have time to train on the noise-added dataset.

8 Discussion

8.1 Reproducibility of our work

Our experiments were performed on Tor Browser 3.6.4 with Tor 0.2.5.7.

We have built a small system that allows WF attacks to operate in the wild. The system takes any full sequence as input, and performs splitting and page identification to identify the web page(s) in the full sequence, if any of the pages is in the monitored set. The reader may download and test our system at https://crysp.uwaterloo.ca/software/webfingerprint/.

Furthermore, to ensure scientific reproducibility of the results in our paper, the following is available at the same URL:

Classifiers. We provide the code for our three split decision classifiers, our three split finding classifiers, and our classification-based and counting-based noise removal algorithms. They include Python and C++ code.

Data. We provide the sensitive and open-world page data set we used (as well as the list of sensitive pages), the multi-page segments we collected for splitting, and the noise we collected for noise removal.

Data collection tools. We provide our data collection tools with instructions on how to use them. These include a modification to Tor to collect ground truth for cell traces and test our WF system above.

8.2 Future work

In this paper, we did not use cell sequences collected in the wild from real clients. While it is technically possible to obtain such cell sequences at a large scale from Tor exit nodes, there are both ethical and legal barriers present; honest Tor exit nodes should not be examining the contents of the traffic passing through them. Furthermore, as it is necessary for us to obtain the client's consent for such data collection, client behavior may change significantly for our experiments, which hurts experimental quality. If these barriers are overcome, our methods could be more thoroughly tested on live Tor clients.

Our work does not propose new page identification techniques, but better page identification techniques can strengthen our algorithms. There is potential for future work in page identification techniques that are resistant to adverse conditions that can negatively affect website fingerprinting, such as a stale training set and background noise.

Unauthenticated
Download Date | 10/23/16 6:26 PM

9 Conclusion

In this work we have tackled three issues in website fingerprinting described by Juarez et al. [13] that separate laboratory conditions and realistic conditions: maintaining a fresh training set, splitting the sequences, and removing noise.

We showed that effective WF training sets can be small enough for an attacker to naively update all data points in a cycle, keeping it fresh. For the splitting problem, we found that splitting based on a time gap of 1 s causes no loss of website fingerprinting accuracy. Previous studies show that the user dwell time has a high probability of being higher than 1 s. We have further demonstrated that a number of machine classifiers can split the sequence with high accuracy when there is no time gap, such as when a user interrupts the loading of one page to load another. To deal with background noise, we proposed several new algorithms, and we found that the attacker can adjust for background noise if he can add the same noise to his data set, although it is hard for the attacker to remove noise accurately.

Overall, we have overcome a number of previously identified barriers to deploying WF in the wild, and demonstrated that even clients using state-of-the-art privacy software like the Tor Browser are subject to snooping by local passive eavesdroppers under realistic conditions.

Acknowledgements

We thank our shepherd Martin Schmiedecker and the anonymous reviewers for their helpful feedback in improving this paper. We thank NSERC for grant RGPIN-341529. 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] 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.

[2] X. Cai, R. Nithyanand, T. Wang, I. Goldberg, and R. Johnson. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *Proceedings of the 21th ACM Conference on Computer and Communications Security*, 2014.

[3] 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.

[4] E. Casalicchio and M. Colajanni. A client-aware dispatching algorithm for web clusters providing multiple services. In *Proceedings of the 10th international conference on World Wide Web*, pages 535–544, 2001.

[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] M. E. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: evidence and possible causes. *Networking, IEEE/ACM Transactions on*, 5(6):835–846, 1997.

[7] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, 2004.

[8] 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.

[9] G. Greenwald. XKeyscore: NSA tool collects 'nearly everything a user does on the internet'. http://www.theguardian.com/world/2013/jul/31/nsa-top-secret-program-online-data, July 2013. Accessed Feb. 2015.

[10] J. Hayes and G. Danezis. k-fingerprinting: a Robust Scalable Website Fingerprinting Technique. arXiv:1509.00789v3, 19 Feb 2016.

[11] 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.

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

[13] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt. A Critical Evaluation of Website Fingerprinting Attacks. In *Proceedings of the 21th ACM Conference on Computer and Communications Security*, 2014.

[14] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas. Circuit fingerprinting attacks: passive deanonymization of tor hidden services. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 287–302, 2015.

[15] 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.

[16] C. Liu, R. White, and S. Dumais. Understanding web browsing behaviors through Weibull analysis of dwell time. In *Proceedings of the 33rd international ACM SIGIR Conference*, pages 379–386, 2010.

[17] 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.

[18] M. Molina, P. Castelli, and G. Foddis. Web traffic modeling exploiting TCP connections' temporal clustering through HTML-REDUCE. *Network, IEEE*, 14(3):46–55, 2000.

[19] A. Panchenko, F. Lanze, A. Zinnen, M. Henze, J. Pennekamp, K. Wehrle, and T. Engel. Website fingerprinting at internet scale. In *Proceedings of the 23rd Network and Distributed System Security Symposium*, 2016.

Unauthenticated
Download Date | 10/23/16 6:26 PM

[20] 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.

[21] M. Perry. A Critique of Website Traffic Fingerprinting Attacks. https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks, November 2013. Accessed Feb. 2015.

[22] 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.

[23] Tor. Tor Metrics Portal. https://metrics.torproject.org/. Accessed Feb. 2015.

[24] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg. Effective Attacks and Provable Defenses for Website Fingerprinting. In *Proceedings of the 23rd USENIX Security Symposium*, 2014.

[25] T. Wang and I. Goldberg. Improved Website Fingerprinting on Tor. In *Proceedings of the 12th ACM Workshop on Privacy in the Electronic Society*, pages 201–212, 2013.

A Split and Noise Removal Features

First, we list the 23 features we used for split finding with kNN. For a candidate cell, we decide if that cell is indeed the correct split by using these features. The feature list is as follows:

1. Five intercell times around the candidate cell. (5)
2. The mean, standard deviation, and maximum intercell time for fifty cells before and after the candidate cell, and the time between the candidate cell and the cell fifty cells before the candidate cell. (4)
3. Time between candidate cell and the next incoming cell. (1)
4. The difference in time between the cell two cells after the candidate cell and the cell two cells before the candidate cell; the cell four cells after and four cells before; and so on, up to eighteen cells. (9)
5. Number of incoming and outgoing cells five and ten cells before and after the candidate cell. (4)

For classification-based noise removal, we used a set of 65 features, as follows:

1. Ten intercell times around the candidate cell. (10)
2. The mean and standard deviation of intercell times for four and twenty-four cells around the candidate cell. (4)
3. Total number of outgoing cells. (2)
4. Directions of all cells within twenty-four cells before and after the candidate cell, including the candidate cell itself. (49)

We include the code for extracting these features in our published data set as in Section 8. The feature set for LF-kNN is very similar to the feature set for kNN, except that it only involved cells in one direction, either before or after the candidate cell.

Unauthenticated
Download Date | 10/23/16 6:26 PM