

On Realistically Attacking Tor with Website Fingerprinting

Tao Wang
Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada
t55wang@cs.uwaterloo.ca

Ian Goldberg
Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada
iang@cs.uwaterloo.ca

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, the training data set is very similar to the testing data set under laboratory conditions, but the attacker may not be able to guarantee similarity realistically. 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 noise, which may adversely affect website fingerprinting, but this effect has not been studied.

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. Although we were not able to remove noise effectively, we will show that it is difficult for users to generate sufficient background noise to disrupt website fingerprinting on Tor. With our techniques, we are able to build the first website fingerprinting system that can operate on packet sequences collected in the wild.

1. INTRODUCTION

In 2009, Panchenko et al. [18] 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, 23] and cutting down the false positive rate (FPR) [22] to practical levels (far below 1%). They have shown that these attacks are computationally cheap and effective in the open-world setting [22]. However, some researchers remain unconvinced that these attacks are effective in the wild [13, 19].

Indeed, the attacks have not been demonstrated to be effective in the wild on Tor; 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 six limiting assumptions:

1. Template websites: Websites can be modeled as templates.

2. 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.
3. 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.
4. Browsing behaviour: Users browse the web sequentially, one page after the other.
5. 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.
6. No background traffic: The adversary can filter out all background traffic.

Amongst those, assumption 1 is only made in one specific work by Cai et al. [3] ¹ Assumption 2 has been dealt with by previous work [18, 22, 23]. For example, the kNN attacker by Wang et al. [22] 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 3 to 6, as follows:

Freshness (assumption 3). 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. Replicability is indeed possible. Nevertheless, we propose and test a scheme that updates the training set more efficiently by scoring each element based on consistency and relevance.

Splitting (assumptions 4 and 5). 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 which may occur sequentially or even in parallel. We demonstrate the effectiveness of time-based splitting and classification-based splitting.

Background noise (assumption 6). We show that noise removal is a difficult problem and we cannot do so accurately, but

¹We should note that assumption 1 is not actually an assumption of the attacker’s capabilities; it is just a model used to improve classification accuracy. Nevertheless Juarez et al. lists it under their set of assumptions, so we include it for the sake of completeness.

we also show that it is very hard to generate sufficient background noise on Tor to disrupt WF due to the design of the Tor Browser.

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 the CrySP site <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 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 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, 9, 11, 12, 14, 16, 18, 20, 22, 23] 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 [10], 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, 14, 20] performed closed-world experiments on simple encrypted channels, and the newest [18, 22] give open-world results on Tor.

2.1 Closed world on encrypted channels

Cheng et al. [5] (1998), Sun et al. [20] (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 [14] (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 rele-

vant 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. [9], 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 [22] 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 bandwidth starved [21], and attacks are harder [11] due to unpredictable network conditions and Tor’s own defenses [22].

Panchenko et al. [18] (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. [9] 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 [23] 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 From closed world to open world

The number of monitored pages used in early works on open-world WF [3, 18, 23] has been too small for implications on real-world practicality. Wang et al. showed [22] that their earlier attack [23] 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. [22] (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, 23].

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 [8]. 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, wire-tapper, 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.

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:

1. Sensitive data set. Contains 40 instances of 120 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. [22]; we updated the page list and re-collected the data set because many of the pages are no longer accessible.
2. Open world data set. Contains one instance each of 5,000 pages from Alexa’s top 10,000 pages. These pages are used as the non-monitored page set to test the false positive rate.

We also collected data sets where multiple pages were accessed at once, and data sets with noise, using the same methodology. These data sets will be described in the relevant sections below.

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. In fact, the attacker only needs to gather data constantly on only a single desktop-class computer. We show the following:

1. Training set size: We empirically determine that a WF attacker can perform WF with a small training set. An effective training set is small enough that it can be kept fresh simply by updating all data points in a cycle on a single desktop-class computer.
2. Training set update: We propose an algorithm to detect and drop bad data points from a training set to keep it fresh. A trivial algorithm would be to drop the oldest data points. We propose using two other metrics for determining which points to drop: consistency and relevance. We show that these methods can be used to update the training set as effectively as simply updating all data points in a cycle.

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. 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 [22]. We evaluate the effect of n_{inst} and n_{nmsite} on TPR and FPR. In Figure 1 we held n_{nmsite} constant and varied n_{inst} , and in Figure 2 we held n_{inst} constant and varied n_{nmsite} . We see that a higher n_{inst} improves TPR but slightly worsens the FPR and a higher n_{nmsite} improves FPR but slightly worsens the TPR. TPR reaches above 70% and FPR reaches below 0.5% at $n_{inst} = 31$ and $n_{nmsite} = 3700$. At this level of accuracy, for $n_{msite} = 100$, the number of web pages the attacker must load is 6800. 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).

We examine the above value of 6800 to determine whether or not it is realistic. In 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 machine. Juarez et al. observed a decrease in accuracy (in a different algorithm) if data was collected 10 days apart, but they did not show any change in accuracy for fresher data. We therefore claim that the attacker can maintain a sufficiently fresh training set with minimal resources. If the attacker wishes to monitor more sites, the required size of the training set would increase correspondingly.

Juarez et al. showed that if the classifier trains on one version of Tor Browser but tests on another, the accuracy may deteriorate. [13]

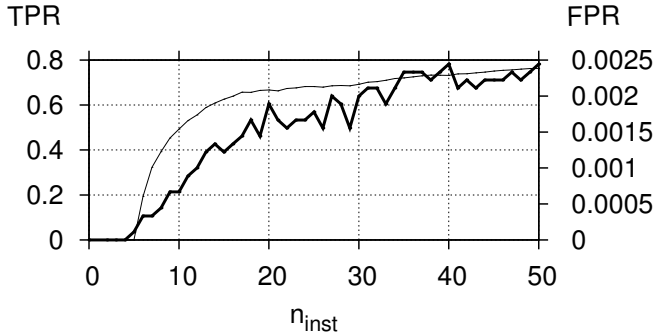


Figure 1: TPR and FPR when n_{inst} varies between 0 and 50, and $n_{nmsite} = 9000$. The thinner line indicates TPR (left y-axis), and the thicker line indicates FPR (right y-axis).

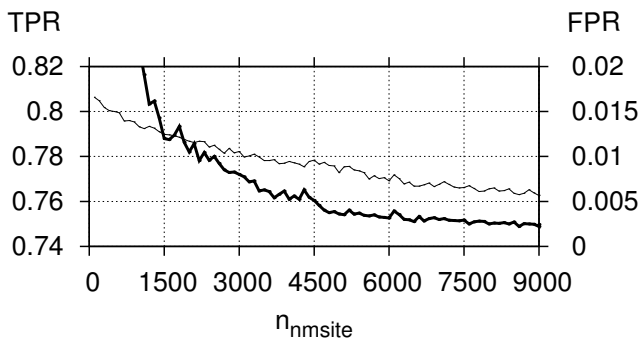


Figure 2: TPR and FPR when n_{nmsite} varies between 0 and 9000, and $n_{inst} = 50$. The thinner line indicates TPR (left y-axis), and the thicker line indicates FPR (right y-axis).

This is because Tor has used two different WF defenses [19], and cell sequences appear different under different defenses. In the worst case, the attacker will have to collect three times as much data to adjust for different versions of Tor Browser, which is still practical. Furthermore, if Tor Browser is outdated, it warns the client in its start page and encourages the client to update.

4.2 Training set update

We showed above that the attacker can maintain a fresh training set by collecting new data and dropping the oldest data points. However, some web pages almost never change while others change daily, so it is inefficient to simply update all data points. In this section, we design a more efficient scheme to maintain a fresh training set.

We assign a score for each point and drop the lowest-scoring data points from the data set. We compute such a score by adding together a *consistency* score and a *relevance* score with random weights, as follows:

1. Consistency: The consistency score is the number of neighbours belonging to the same class as the point itself.
2. Relevance: The relevance score is the number of points from any class that have this point in its top-20 closest neighbour set.

We chose random weights to show that both of these scores are indeed useful, and they can be combined in any way; we did not overfit our score.

Table 1: TPR of kNN after updating the training set using our scheme with relevance and consistency scoring (Re-con), and several other baseline comparisons (Stale, Random, Fresh).

Update method	Accuracy (TPR)
Stale	0.711 ± 0.001
Random	0.756 ± 0.005
Fresh	0.772 ± 0.001
Re-con	0.775 ± 0.003

To test if our score is effective for updating the whole training set, we construct the testing set by including only the freshest half of the original data set, and we construct the training set by including the highest-scoring half of the original data set. We use leave-one-out cross validation, so that while testing an element from the testing set, if it also appears in the training set, we do not allow the classifier to use that element in the training set. If classification of the testing set is accurate with such a training set, then the score is effective. We compare our score with several baselines:

1. Stale: The training set is the least fresh half of the original data set. The attacker does not update the training set.
2. Random: The training set is randomly selected. The attacker updates random points in the training set.
3. Fresh: The training set is the freshest half of the original set (as is the testing set). The attacker updates all points in the training set.

The Stale training set is older than the Fresh training set by about a week. We show the results in Table 1 by generating the training sets 100 times with the above methods and classifying the testing set with the generated training set. We show only the TPR as there was no significant change in the FPR. Our new scheme is listed as Re-con (relevance and consistency) in the table. The low standard deviation of the accuracy value convinces us that our updating scheme is significantly more useful for classification than simply a trivial random scheme or the Stale training set, and it is about as useful for classification as the Fresh training set (which is the most expensive to maintain). Therefore, the attacker can indeed maintain an effective training set without having to update all data points.

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. Therefore, these attacks cannot operate in the wild unless we can split accurately without this ground truth.

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 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 3:

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 seconds t_{gap} , 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_{gap} . 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 beyond 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,

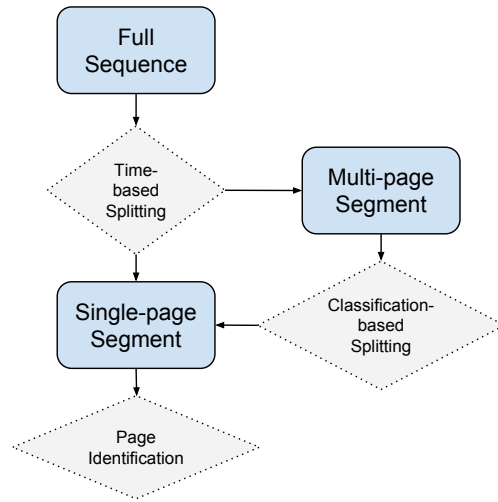


Figure 3: Splitting process, with ideal results. Overall, the attacker wants to obtain single-page segments from the full (raw) sequence.

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, we show in Section 6.2 that they are unlikely to occur, and 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_{gap} . Our choice of t_{gap} seeks to minimize the chance of splitting single-page segments, which should not be split any further. A larger t_{gap} reduces such a risk but renders multi-page segments more likely if the client’s *dwelling time* is small.² To split these we need to apply classification-based splitting, which we discuss in the next section.

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

1. Splitting a single-page segment with t_{gap} . 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_{gap} . 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_{gap} increases the chance of the former and decreases the chance of the

²The dwell time is the amount of time a user stays on a page between two page visits.

latter, and vice-versa, so a suitable value must be chosen. We will show how we choose t_{gap} 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_{gap} 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 machine 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 [22], 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 [18]. 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 machine 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 machine 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. [22] 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. 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*.
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 [14] 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 intentionally chose methods similar to ones that succeeded for WF as WF is similar to splitting. 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 $t_{gap} = 1.5$ 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 t_{gap} .
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.³ 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 of 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 some value t_{gap} , then we split the sequence there.

A smaller t_{gap} has a larger chance of splitting single-page segments, which is erroneous. We measured this effect by applying time-based splitting to single-page segments, and testing page identification with kNN on the resulting segments while varying t_{gap} from 0.2 to 4 seconds. If t_{gap} 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. In Figure 4 we plot the resultant negative effects on the true positive rate (there was no noticeable effect on the false positive rate). We see from Figure 4 that the drop in accuracy becomes negligible after around $t_{gap} = 1.5$ s. We therefore suggest $t_{gap} = 1.5$ s for time-based splitting.

It may be argued we should reduce t_{gap} below 1.5 s, to trade an increased chance of splitting single-page segments for an increased chance of splitting multi-page segments. Recall that classification-based splitting does not handle segments with three or more pages (two or more dwell times less than t_{gap} in a row), so we want them to be rare after time-based splitting.

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],

³Any local, passive attacker can use our splitting method from `tcpdump` info; we only need cell data for experimental ground truth.

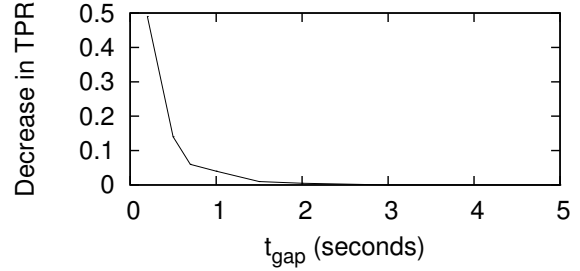


Figure 4: Decrease in TPR of the kNN classifier when splitting cell sequences with t_{gap} .

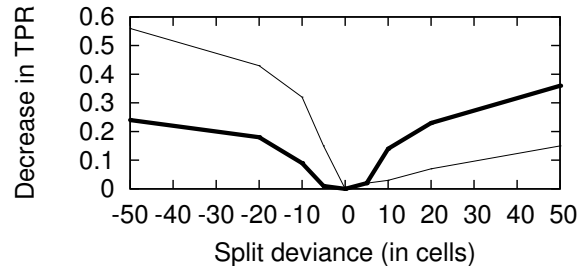


Figure 5: Decrease in TPR if the split is guessed incorrectly. A negative split deviance indicates that the guessed split was before the correct split, and a positive split deviance indicates it was after. There are two lines: The thicker line is for the segment that received extra cells because the split was in the wrong place; the thinner line is for the segment that lost cells. The false positive rate did not change significantly.

lognormal [17] and “negative-aging” Weibull [15, 17] distributions with the mean being around a minute. Therefore, the probability of the dwell time being under $t_{gap} = 1.5$ s twice in a row (for multi-page segments with three or more pages) is small. In other words, reducing t_{gap} further would produce no notable benefit but incur a heavy cost in page identification accuracy (as in Figure 4).

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 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 split accuracy by randomizing parts 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 seg-

Table 2: 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

ment, 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 5. We see that under 5 cells the 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.

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 and 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 4 in Appendix B.

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 5 in Appendix B.

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 2. The table shows that classification of Class 3 is indeed the most difficult: in fact, we cannot expect to correctly identify overlapping pages.

7. REMOVING NOISE

In this section, we will discuss another aspect of realistic cell sequences that may deteriorate theoretical accuracy values: background noise. The client may, for example, download a file, listen to music, or watch a video while browsing. These activities are per-

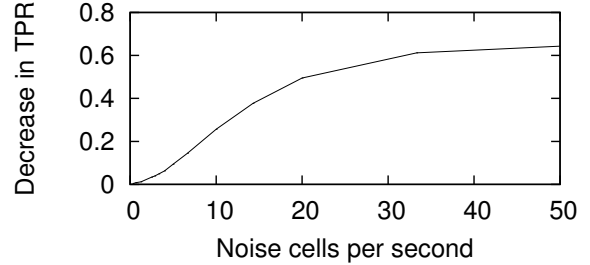


Figure 6: Decrease in TPR of the kNN classifier when random noise is added.

sistent and may interfere with both splitting and page identification if the noise transmission rate is high enough. We characterize noise in Section 7.1. Then, we present two negative results:

1. Removing background noise is difficult (Section 7.2). We demonstrate the difficulty both practically and analytically. Practically, we will describe how our classification-based and counting-based approaches both failed to remove noise accurately. Analytically, we will show that there are inherent difficulties in removing this noise due to variation in intercell timing.
2. Generating noise is difficult with Tor Browser (Section 7.3). Due to the design of Tor Browser, it is difficult to generate sufficient noise on Tor to disrupt WF, as Flash is disabled and file downloading is slow and discouraged.

7.1 Characterizing noise

We first characterize noise by identifying the noise rate (measured in cells per second) that is necessary for WF accuracy to drop. To do so, we add random cells to cell sequences and ask the kNN classifier 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. 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 6. The figure shows that decrease in TPR becomes significant after 5 cells per second, and the TPR has dropped significantly at around 20 cells per second (80 kbps). In general, video streaming, audio streaming and file downloading may exceed this bit rate, while other sources of noise such as chatting and AJAX may not.

7.2 Difficulties in removing noise

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. Second, we downloaded 10 minutes of audio from a music site. When plotting the number of cells loaded over time, our audio file was shaped as a step function, whereas our file download was continuous.

We merged these noise cells with web cells (keeping timing and ordering), and attempted to remove only the noise cells. We attempted two approaches to noise removal: the classification-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.

With our noise data, we will find that there are inherent difficulties in both approaches due to the high degree of randomness in noise.

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. Although we cannot prove that noise removal is *impossible*, at least it is significantly more difficult than splitting and WF. For this experiment, we reduced the file download rate to the web browsing rate, so that the number of cells of each class is about equal.

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 the SVM because its kernel method may be able to find an implicit feature space where the features are usable, whereas kNN has no such capability.

There were two classes: web cells and noise 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 $67 \pm 10\%$ for both classes. These values are low compared to what is necessary for accurate page identification after noise removal. Figure 6 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 web cell rate, we found that the classification accuracy needs to be above 92% for the kNN 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.

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 web cells and noise cells together, as above.

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 3. 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 d/l noise and 42% for audio noise if cells were simply removed randomly, at a rate proportional to the total number of web cells and noise cells. The noise-removed sequence had, on average, 20% total length difference from the true sequence, compared to around 40% if no noise removal was done at all. Results from both counting-based algorithms are unrecognizable by the kNN page identification classifier (accuracy is close to random guessing).

Table 3: Noise removal accuracy for our counting-based algorithm. Noise and web 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	Web cells removed
File d/l	$55\% \pm 16\%$	$39\% \pm 21\%$
Audio	$66\% \pm 14\%$	$32\% \pm 12\%$

Interestingly, we found that the kNN page identification classifier still had a true positive rate over 50% if it was also allowed to train on the results of noise removal; however, this is expensive as the attacker would have to prepare a large number of noise data sets for each possible source of noise. Nevertheless, this is not impossible.

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 inter-cell 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 remove a web 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 d/l 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.

7.3 Difficulties in noise generation

Although our results above show that noise removal is difficult, we found that it may be unlikely for Tor users to generate sufficient noise specifically on Tor Browser. The reasons are as follows:

1. *Flash is disabled.* Most multimedia sources require Flash, which is disabled on Tor Browser. On some (but not all) of these pages, a browser pop-up will ask if the user wants to enable Flash. If the pop-up does not appear, the user cannot view the page. Sometimes, even if the user clicks Allow on the pop-up, the page will still fail to load even after refreshing. It is a hassle to load Flash pages on Tor Browser. A mitigating factor to this difficulty is that YouTube recently started using HTML5 by default for many of its videos, which may allow Tor users to view YouTube videos more easily.
2. *Low bandwidth.* Tor nodes offer widely varying bandwidths, which may sometimes make it difficult to load multimedia even at the lowest quality. For example, YouTube suggests a 1,000 kbps connection to view its 360p videos, but Tor circuits often offer less bandwidth [21].
3. *Blocking due to localization.* Some multimedia sites, such as Netflix and Spotify, do not serve users in many countries.

Since Tor users appear to originate from their exit node and thus their exit node’s country, these sites may not be accessible on Tor. This is especially problematic for multimedia sites due to copyright issues.

4. *Blocking Tor.* Some sites, such as Twitch, block Tor users flat out by accessing the public list of Tor relays and blocking those IPs. This allows sites to ban misbehaving users, and in recent years this has become a growing problem for Tor [7].

These issues suggest that the average Tor Browser user is not likely to generate significant noise in their web browsing to adversely affect WF.

Another reason why Tor users might not generate significant noise to affect WF is Tor’s circuit dirtiness mechanism. In Tor, circuits do not (by default) accept new TCP connections more than 10 minutes old. Noise that lasts longer than 10 minutes on a single TCP connection (for example, a file download) may cease to be noise at all if the attacker is able to distinguish among different TCP connections. There are two cases where the attacker can distinguish between TCP connections: first, if the client randomly chooses a different entry guard (by default clients have three entry guards to choose from when establishing circuits); second, if the attacker *is* the entry guard currently used by the client.

8. DISCUSSION

8.1 Reproducibility of our work

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 from the CrySP site <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 cell traces and test our WF system above.

8.2 Future work

In this paper, we do not claim that WF is now truly practical, although we have tackled three important barriers that lay in the way of practicality. This is because we do not have cell sequences collected in the wild from real clients, so we cannot know the final WF accuracy. 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. We hope to engage the community in a discussion of this issue.

While we have built a system to test WF attacks in the wild, the training set in our system is not actively updated. Although we

have demonstrated that it is indeed practical to keep the training set fresh in Section 4, we have not done so yet. We wish to populate our training set with cell sequences gathered from real clients, but we currently cannot do so.

9. CONCLUSION

In this work we have tackled three issues in website fingerprinting that separate laboratory WF and realistic WF: 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. The attacker can maintain the training set even more efficiently by using scoring metrics such as consistency and relevance; using these metrics was as helpful for classification as simply updating all data points.

For the splitting problem, which is necessary for website fingerprinting to operate in the wild, we found that splitting based on a time gap of 1.5 s causes no loss of website fingerprinting accuracy. Previous studies show that the user dwell time has a high probability of being higher than 1.5 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 loads one page right after another.

We found that noise removal is difficult for the attacker. Our classification-based and counting-based algorithms both failed to remove noise accurately, and even a small amount of error in noise removal translates to a large error in page identification. However, we also find that users are unlikely to generate sufficient noise on Tor Browser to disrupt WF, because of practical limitations of Tor.

Overall, we have built a system that can perform WF on sequences taken in the wild, although the final WF accuracy in the wild is yet unknown because there are ethical and legal barriers to data collection in the wild.

10. 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. A call to arms: Helping Internet services accept anonymous users. <https://blog.torproject.org/blog/call->

- arms-helping-internet-services-accept-anonymous-users, August 2014. Accessed Feb. 2015.
- [8] R. Dingleline, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, 2004.
- [9] 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.
- [10] 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.
- [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] 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] 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.
- [16] 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.
- [17] 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.
- [18] 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.
- [19] M. Perry. A Critique of Website Traffic Fingerprinting Attacks. <https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks>, November 2013. Accessed Feb. 2015.
- [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 Feb. 2015.
- [22] 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.
- [23] 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.

APPENDIX

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.

B. SPLITTING RESULTS

In this section we present the full tables of results from splitting. Table 4 shows the results of split decision, and Table 5 shows the results of split finding.

Table 4: 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

Table 5: 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)