A Critical Evaluation of Website Fingerprinting Attacks

Marc Juarez[1], Sadia Afroz[2], Gunes Acar[1], Claudia Diaz[1], Rachel Greenstadt[3]

[1]KU Leuven, ESAT/COSIC and iMinds, Leuven, Belgium
{name.surname}@esat.kuleuven.be

[2]UC Berkeley
sadia.afroz@berkeley.edu

[3]Drexel University
greenie@cs.drexel.edu

ABSTRACT

Recent studies on Website Fingerprinting (WF) claim to have found highly effective attacks on Tor. However, these studies make assumptions about user settings, adversary capabilities, and the nature of the Web that do not necessarily hold in practical scenarios. The following study critically evaluates these assumptions by conducting the attack where the assumptions do not hold. We show that certain variables, for example, user's browsing habits, differences in location and version of Tor Browser Bundle, that are usually omitted from the current WF model have a significant impact on the efficacy of the attack. We also empirically show how prior work succumbs to the base rate fallacy in the open-world scenario. We address this problem by augmenting our classification method with a verification step. We conclude that even though this approach reduces the number of false positives over 63%, it does not completely solve the problem, which remains an open issue for WF attacks.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; K.4 [**Computers and Society**]: Public Policy Issues—*Privacy*

Keywords

Website fingerprinting; Tor; privacy

1. INTRODUCTION

Anonymous communication systems are designed to protect users from malicious websites and network eavesdroppers by providing means to hide the content and metadata of communications. *The Onion Router* (Tor), with about three million daily users, is the most popular anonymous communication network. It is specially designed for low-latency applications such as web browsing [8, 29]. Tor routes connections through three-hop *circuits* and encrypts the traffic in layers using *onion routing* [10], so that none of the relays can know both the origin and the destination of the communication at the same time.

Although Tor hides the routing information and communication content, the analysis of the network traffic alone may provide very rich information to an attacker with sufficient capabilities. Using timing, frequency and length of the messages, an attacker can bypass otherwise very robust security mechanisms and identify the communicating parties [7, 22].

Website Fingerprinting (WF) allows an adversary to learn information about a user's web browsing activity by recognizing patterns in his traffic. The adversary in this attack compares the network traces of Tor users to a set of prerecorded webpage fingerprints to identify the page that is being accessed. WF is different from traffic correlation attacks where the adversary has access to both entry and exit nodes and matches the patterns in the incoming and outgoing traffic to the Tor network [14]. WF is also different from deep packet inspection protocols and related traffic analysis techniques that are used to censor Tor [13].

Several previous works demonstrate the effectiveness of WF attacks on Tor despite the encryption, padding and application level defenses such as randomized pipelining [3, 11, 23, 26, 32]. Although we appreciate the importance, novelty and scientific rigor of these studies, the assumptions they made vastly simplify the problem and give unrealistic advantages to the adversary by either simplifying the world or overestimating the adversary's capabilities. Some of these assumptions are challenging and hard to attain realistically in practice [11]. For example, most current works implicitly/explicitly assume that the adversary and user both use the same Tor Browser Bundle (TBB), visit the same localized version of a limited set of pages/sites almost at the same time (or a few days apart) by using only one tab browsing. However, violating at least one of these assumptions can reduce the efficacy of the attack significantly to a point that might not make WF a threat in the real world. The authors of these studies aim to provide an upper bound for the efficacy of the attacks and argue that a particular attacker or scenario might satisfy them. Also it has been argued that in a real-world scenario, proposed countermeasures against WF would actually be more efficient than these studies have estimated [25].

CCS'14, November 3–7, 2014, Scottsdale, Arizona, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2957-6/14/11 ...$15.00.
http://dx.doi.org/10.1145/2660267.2660368.

The goal of this study is to assess the practical feasibility of WF attacks proposed in prior work. Our contributions and their organization in the paper are as follows:

A critical evaluation of assumptions made by prior WF studies: We provide an extensive model of the WF attack, define the assumptions made by prior WF studies on the adversary, the client-setting and the Web. We argue that these assumptions are unrealistic because they are oversimplifying the problem thus are unlikely to hold in practice (Section 3).

An analysis of the variables that affect the accuracy of WF attacks: We pin down the variables that were omitted from the models considered in previous work that have an impact on the practical effectiveness and feasibility of the attacks (Section 4). We present the results of a set of comparative experiments to evaluate the effects of these variables on traffic traces and classifier accuracy. We show that, for some of the variables, the accuracy can drop up to 70%.

An approach to reduce false positive rates: We show the effect of false positives in an open-world of 35K webpages and use *Classify-Verify* in the WF domain on the estimated probabilities of the classifier which reduces the number of false positives over 63% (Section 5).

A model of the adversary's cost: We model the cost that an adversary would incur to maintain a successful WF system (Section 6). We suspect that maintaining a perfect WF system is costly as the adversary needs to collect information about different localized versions of the webpages, user's browsing settings and update the system over time to recover from data staleness.

2. WEBSITE FINGERPRINTING

The main objective of an adversary in a typical WF scenario is to identify which page the user is visiting. The adversary may want to learn this information for surveillance or intelligence purposes.

The WF attack is typically treated as a classification problem, where classification categories are webpages and observations are traffic traces. The adversary first collects traffic traces by visiting webpages and trains a supervised classifier using features such as the length, direction and inter-arrival times of network packets.

Whenever a user visits a webpage over Tor, the adversary records the network trace by, for instance, intercepting the traffic locally (LAN), by having access to routers of the user's ISP, or by controlling an entry guard to the Tor network. He then runs the classifier on the intercepted network trace to guess the site the user has visited.

The first WF attacks were developed to identify pages within a single website over SSL connections [4, 20]. In 2002, Sun et al. tackled the more challenging problem of identifying individual pages within a set of websites [28] which led to Hintz's attack on an anonymizing web proxy (*SafeWeb*) [12]. Many WF studies on one-hop proxy attacks have followed [2, 9, 16, 18].

Herrmann et al. [11] deployed the first WF attack on the Tor anonymity network with only a 3% success rate for a world of 775 pages. The attacks that followed significantly improved the accuracy: Shi and Matsuura obtained 50% success rate for 20 pages [26]; Panchenko et al., obtained 54.61% accuracy using Herrmann's dataset [23]; and, finally, Cai et al. and Wang and Goldberg report success rates over 90% using edit-distance based classifiers on a world of 100 pages [3, 32].

3. MODEL

We model the WF adversary as *passive* and *local*: the adversary is able to eavesdrop on the user's traffic, but cannot add, drop or modify packets. We also assume that the adversary cannot decrypt the contents of the network packets, as that would render WF attacks unnecessary.

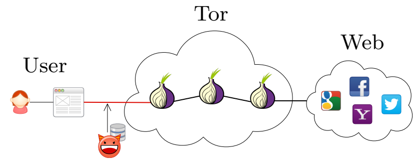
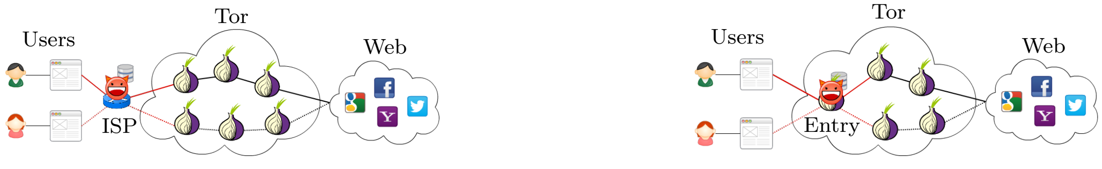


Figure 1: The basic WF targeted attack in Tor.

Figure 1 depicts the basic WF scenario: the attacker taps the network between the victim and the Tor entry guard and collects traffic traces, which he then compares against his database of webpage fingerprints. We make a distinction between two types of attacks based on the number of users targeted by the adversary and the resources at his disposal.

Targeted: In this attack, the adversary targets a specific victim to retrieve his browsing activity. This allows the attacker to train a classifier under conditions similar to those of the victim (see Figure 1), potentially increasing the success of the attack. The adversary may have enough background knowledge about the user to reproduce his configuration, or he could detect it from the observed traffic data. In Section 6, we discuss how difficult it is for the attacker to discover properties about the user's setting.

Non-targeted (dragnet surveillance): In this case, the adversary targets a set of users instead of one. ISPs, Internet exchanges and entry guard operators are in a position to deploy this attack since they can intercept the network traffic of many users (see Figures 2a and 2b, respectively). The attacker trains the classifier on a specific setting and uses the same classifier on all communications that he observes.



(a) ISP level adversary.

(b) Malicious entry guard.

Figure 2: WF Non-targeted attacks in Tor.

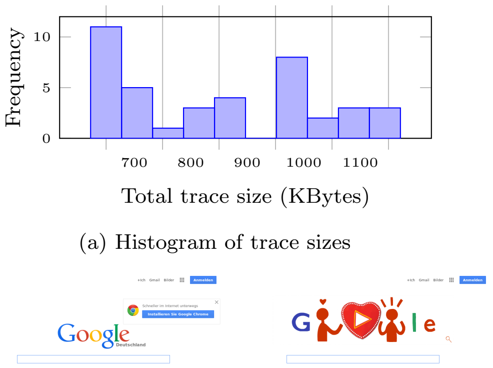
3.1 Assumptions

We compiled the assumptions made in the literature of WF attacks on Tor. We divided the basic model in three parts: (i) Client-side, (ii) Adversary, and (iii) Web, and classified the assumptions according to the part of the model they relate to. We note that the assumptions are not mutually exclusive and are open to other classifications. The papers that explicitly mention these assumptions are listed in Table 1.

Client-setting.

Closed-world: *There are only k webpages that the user may visit.* This is a very strong assumption because k is always very small compared to the actual number of existing webpages. Some authors have also evaluated their classifiers in an *open-world* scenario [3, 23, 32], where there is a set of k target pages being monitored but the user is allowed to visit pages that are not in that set.

Browsing behaviour: *The users follow a specific behaviour.* For example: users browse the web sequentially, one page after the other, having only a single tab open. Nevertheless, real-world studies found that users tend to have multiple open tabs or windows [21, 31], which allow them to load several pages at once. Although we do not have access to data collected from Tor users, it is safe to think that Tor users exhibit this behavior since their connection is slower.



(a) Histogram of trace sizes

(b) Doodle and standard versions

Figure 3: Different versions of the Google homepage and corresponding histogram of trace sizes for 40 visits. The histogram shows that the distributions of packet sizes are significantly different for the same page visited on different dates.

Web.

Template websites: *All websites are built using templates.* Cai et al. used a *Hidden Markov Model* (HMM) to leverage the link structure of websites for WF [3]. The authors made this assumption in order to simplify their model and reduce the number of states in the HMM.

There are further unrealistic assumptions about the Web that are not explicit but that may improve the accuracy of the WF attack. For instance, one recent study used localized (German) versions of the webpages in order to avoid different language versions [32]. In our experiments, we observed cases such as *ask.com* where the total trace size of the English version was about five times bigger than the German version of the same webpage. Given that the language of the webpage will be selected according to the Tor exit node, assuming that users would visit the same local version is a clear advantage to the adversary.

Even if we limit ourselves to localized versions of webpages, there are still other sources of dynamism such as bot detection, changing graphics and third-party content. Figure 3a shows the histogram of sizes of 40 traces collected from *google.de* between February 14-16. We observe a clear distinction between two sets of trace sizes. The group on the left corresponds to the page without a doodle (left in Figure 3b). The group on the right with larger network footprint corresponds to the version with a special doodle for St. Valentine's day (right in Figure 3b). Note that Wang and Goldberg concluded that sites that change in size are hard to classify correctly [32].

Adversary.

Page load parsing: *The adversary can detect the beginning and the end of different page loads in a traffic trace.* This has been shown to be a very hard task in the context of session reconstruction from real-world network traffic [6].

No background traffic: *The adversary can filter all background network traffic produced by other applications or other connections going through the same Tor circuit.* Tor is increasingly used in settings where multiple applications or complete operating system traffic is sent over the Tor network[1]. In these cases, separating the browsing traffic from the background traffic may present a nontrivial challenge to the adversary.

Replicability: *The adversary can train his classifier under the same conditions as the victim.* For instance, the adversary is assumed to be able to replicate client-side settings such as operating system, network connection or Tor Browser Bundle (TBB) version. This assumption allows researchers to train and test on data collected using the same settings. Depending on the type of attack this may be impossible, as the adversary may encounter difficulties in detecting and replicating users' configuration (especially in non-targeted attacks).

Assumption	Explicitly made by
Closed-world	[11, 26]
Browsing behavior	[11]
Page load parsing	[3, 11, 23, 26, 32]
No background noise	[3, 11, 23, 26, 32]
Replicability	[11, 26]
Template websites	[3]

Table 1: Assumptions and references to papers that make explicit mention to them.

4. EVALUATION

In this section we challenge some of the assumptions described in Section 3. The assumptions are: Closed-world, browsing behavior, no background traffic and replicability.

[1] `https://tails.boum.org/`

For each assumption we identify the variables that are ruled out of the model by the assumption. Our objective is to measure the effect of these variables on the accuracy of WF attacks.

4.1 Datasets

We used two different lists of URLs for our crawls: the top Alexa ranking and the Active Linguistic Authentication Datatset (ALAD) [15]. The Alexa dataset is a well known list of most visited URLs that has been widely used in previous WF studies as well as in other research domains. Testing on data collected by crawling URLs in the Alexa ranking implies that the adversary always knows which pages the users are going to visit and can train a classifier on those pages. We want to test the reliability of a classifier when this assumption does not hold.

The Active Linguistic Authentication Dataset (ALAD) [15] is a dataset of web visits of real-world users, collected for the purpose of behavioral biometrics evaluation. The complete dataset contains data collected from 80 paid users in a simulated work environment. These users used the computers provided by the researchers for a total of 40 hours to perform some assigned tasks: open-ended blogging (at least 6 hours a day) and summary writing of given news articles (at least 2 hours a day). We found that these users were browsing the web to check their emails, social network sites and searching for jobs.

Top Alexa	Home page	Other pages	Not in Alexa
100	4.84%	50.34%	44.82%
1000	5.81%	60.26%	33.93%
10000	6.33%	68.01%	25.66%

Table 2: ALAD dataset statistics

The users browsed total 38,716 unique URLs (excluding news articles) which were loaded 89,719 times. These URLs include some top ranked Alexa sites, such as `google.com`, `facebook.com` and `youtube.com` and some sites that are not in Alexa top 1 million list, such as `bakery-square.com`, `miresearch.org`. Over 44% of the sites were not in the Alexa top 100, and 25% of the sites were not in the Alexa top 10000. Over 50% sites were pages other than the front page (shown in Table 2), such as different search pages on Google/Wikipedia, a subdomain of a site (such as `dell.msn.com`) and logged-in pages of Facebook.

4.2 Data collection

To collect network traces, we used TBB combined with Selenium[2] to visit the pages, and recorded the network packets using a network packet dump tool called `dumpcap`. We used the Stem library to control and configure the Tor process [30]. Following Wang and Goldberg we extended the 10 minute circuit renewal period to 600,000 and disabled the `UseEntryGuards` to avoid using a fixed set of guard nodes [32]. We also used their methods to parse the Tor cells and remove noise by filtering acknowledgements and SENDMEs.

We crawled the webpages in *batches*. For each batch, we visited each page 4 times and collected between 5 and 10 batches of data in each crawl, resulting in 20 to 40 visits for each webpage in total. We waited 5 seconds after each page had finished loading and left 5 second pauses between each visit. The batches are collected in a round-robin fashion, hours apart from each other. We made over 50 such crawls for the experiments presented in this paper and we will share the data with other researchers upon request.

We used two physical and three cloud-based virtual machines to run crawls from different geographical locations. In order to have identical crawler configurations, we used Linux Container (LXC) based virtualization running on the same distribution and version of the GNU/Linux operating system. We disabled operating system updates to prevent background network traffic and never ran more than one crawler on a machine at the same time. We made sure that the average CPU load of the machines is low, as this may affect the WF defenses shipped in the TBB[3].

4.3 Methodology

In order to reduce the confounding effect of other variables in the measurement, we crawled the same set of webpages multiple times by changing the value of the variable under evaluation and fixing the rest of the variables. For each variable that we wanted to evaluate, we defined a *control crawl* by setting the variable to its default value (e.g., `UseEntryGuards` = 1), and a *test crawl*, by setting the variable to the value of interest (e.g., `UseEntryGuards` = 0).

Note that the randomized nature of the path selection algorithm of Tor and effect of time are two control variables that we cannot completely fix when measuring the effect of other variables. We tried to overcome this by using cross-validation and minimizing the time gap between the control and test crawl in all of our experiments.

These *experiments* are composed by the two following steps:

1. k-fold cross-validation using data of the control crawl.
2. Evaluate classifier's accuracy training on the control crawl and testing with data from the test crawl.

The accuracy obtained in Step 1, the case in which the adversary can train and test under the exact same conditions, is used as a baseline for comparison. We then compare the accuracy obtained in Step 2 with this baseline. We specify the details of the cross-validation in Step 1 and the testing in Step 2 later in this section.

Classifiers designed for WF attacks are based on features extracted from the length, direction and inter-arrival times of network packets, such as unique number of packet lengths or the total bandwidth consumed. The variables we evaluate in this section affect traffic features and therefore may affect each classifier in a different manner (cf., [33]). For the present study we tried to pick a classifier for each of the learning models and sets of features studied in prior work. In Table 3 we list the classifiers that we have evaluated.

We observed that the relative accuracy changes are consistent across the classifiers and the variables we evaluated. In most cases, classifier `W` performed better than the others. For this reason, our presentation of the results is focused on classifier `W`.

Classifier `W` is based on the *Fast Levenshtein-like* distance [32]. We used this classifier instead of the one based on the *OSAD*[4] distance presented in the same paper. Although

[2] `http://docs.seleniumhq.org/`

[3] `https://trac.torproject.org/projects/tor/ticket/8470\#comment:7`

[4] Optimal String Alignment Distance

Name	Model	Features
H [11]	Naive Bayes	Packet lengths
P [23]	SVM	Packet lengths Order Total bytes
D [9]	N-grams	Total time Up/Downstream bytes Bytes in traffic bursts
W [32]	SVM (Fast-Levenshtein)	Cell traces
T	Decision tree	Same features as P

Table 3: Classifiers used for the evaluation.

the latter attained greater accuracy in Tor, it is considerably slower and can become impractical in certain circumstances, as we will further analyse in Section 6. Furthermore, for the OSAD distance we obtained over 90% accuracy scores in our control crawls and, as in the original paper, we consistently observed a 20% decrease in accuracy when evaluating classifier W. For this reason, we believe the results obtained with this classifier are comparable with its more accurate counterpart.

For the evaluation of each classifier we followed a similar approach to the one described by Dyer et al. First, we fixed the size of the world to a certain number of pages (k). For each page of the training crawl, we selected n_{train} random batches and picked T_{train} traffic traces in total. For each page of the testing crawl, we selected n_{test} random batches and picked T_{test} traces in total. We averaged the results by repeating each experiment m times, each time choosing a different training and testing set. Then, the accuracy of the classifier was calculated by $\frac{Total\ correct\ predictions}{Total\ test\ instances} = \frac{p}{m\,T_{test}}$, where p is the total number of correct predictions. We made sure that for the validation of the control crawl, training and testing traces were never taken from the same batch. The classification parameters are listed in Table 4.

Parameter	Description
k	Number of sites in the world.
$n_{train/test}$	Number of batches for training/testing.
$T_{train/test}$	Number of instances for training/testing.
p	Total number of predictions.
m	Number of trials.

Table 4: Description of classification parameters defined for an experiment.

From now on, we refer to *Acc control* as the average accuracy obtained for the m trials in control crawl (Step 1) and *Acc test* as the average accuracy for m trials obtained in Step 2. In the results, the standard deviation of the accuracy obtained for m trials is shown in parentheses next to the average value.

We also have designed our own attack based on decision tree learning and using the features proposed by Panchenko et al. [23]. Decision tree learning uses binary trees as data structures to classify observations. Leaves represent class labels and nodes are conditions on feature values that divide the set of observations. Features that better divide the data according to a certain criterion (*information gain* in our case) are chosen first.

For these experiments we have extended Dyer et al.'s Peekaboo framework [9] and the source code of the classifier presented by Wang and Goldberg [32].

4.4 Time

Webpages are constantly changing their content. This is reflected in the traffic traces and consequently in the accuracy of the WF attack. In this section we used crawls of the Alexa Top 100 pages taken at different instants in time to evaluate the effect of staleness on WF.

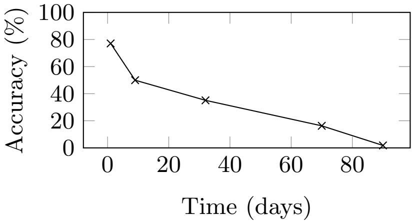


Figure 4: Staleness of our data over time. Each data point is the 10-fold crossvalidation accuracy of the classifier W which is trained using the traces from $t = 0$ and tested using traces from $0 \leq t \leq 90$. For every experiment $m = 10$, $n_{train} = 9$, $n_{test} = 1$, $T_{train} = 36$, $t_{test} = 4$ and $k = 100$.

For this experiment, we train a classifier using the traces of the control crawl, collected at time $t = 0$, and test it using traces collected within 90 days of the control crawl ($0 \leq t \leq 90$). Figure 4 shows the accuracy obtained for the W classifier for crawls over the same list of URLs at different points in time. For the rest of the classifiers we observed similar drops in accuracy. The first data point is the accuracy of the control crawl, taken at $t = 0$. The second point is taken 9 days after the control crawl ($t = 9$), and so on until the last data point that corresponds to a test crawl taken 90 days later.

We observe that the accuracy drops extremely fast over time. In our experiments it takes less than 10 days to drop under 50%. Since we observed such a drop in accuracy due to time, we picked control and test crawls for the following experiments that fall within a period of 5 days.

This strong effect of time in the classifier's accuracy poses a challenge to the adversary who will need to train the classifier on a regular basis. Depending on the size of the world that he aims to cover, the cost of training may exceed the time in which data provides reasonable accuracy rates. We discuss this point in more detail in Section 6.

4.5 Multitab browsing

In this experiment we evaluate the success of a classifier when trained on single tab browsing, as in prior WF attacks, and tested on traces collected with multitab browsing.

In order to simulate multitab browsing behaviour, we crawled the home pages of Alexa Top 100 sites [1] while loading another webpage in the background. The background page was loaded with a delay of 0.5-5 seconds and was chosen at random from the same list, but kept constant for each batch. Then we train five classifiers from prior work, P, H, D, W, T (described in Table 3), using single tab traces of Alexa Top 100 webpages and test it using the multitab traces we collected. We consider a classifier successful if it can identify either the foreground page or the background page.

We observe a dramatic drop in the accuracy for all the classifiers with respect to the accuracy obtained with the control crawl (when the classifiers are trained and tested using single tab traces), even when the delay between the first

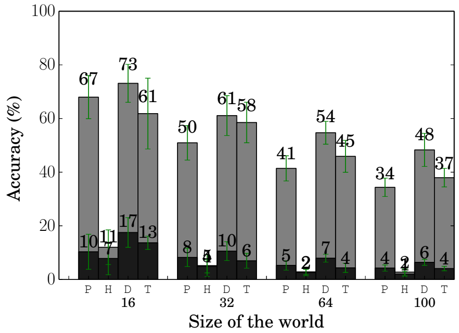


Figure 5: Average accuracies of P, H, D, T classifiers for a delay of 0.5 sec between the loading start times of the foreground page and the background page. Light gray bars represent the accuracies of the control crawls (Step 1). We plot in darker colour the accuracies obtained by training in the control and testing in the multitab crawl (Step 2). Green intervals indicate the standard deviation of the accuracy.

page and the background page was of 0.5 seconds (Figure 5 and Table 5). We also observe a drop in the accuracy while we increase the size of the world, although the change in the accuracy was similar for all classifiers ((Figure 5).

We notice very similar accuracies for classifiers P and T in this experiment. These two classifiers are built using the same set of features but different learning models. This might imply that the specific learning model is not as important for a successful attack as the feature selection. Dyer et al. reported a similar observation between Naive Bayes and SVM classifiers.

We also vary the time gap between the two pages to account for different delays between opening the two tabs. Since the W classifier is based on an edit-distance, we expect the distance between the observed traffic trace and the trace of any of the two loaded pages to be smaller with respect to shorter delays, since there would be less overlap between the traffic traces of the two loaded pages. However, we do not observe a significant evidence that may support this hypothesis in the evaluation for 0.5, 3 and 5 seconds of delay (Table 5) . The average page load for the test crawl for the 5 second gap experiment is 15 seconds, leaving on average 30% of the original trace uncovered by the background traffic. Even in this case, the accuracy with respect to the control crawl drops by over 68%.

Delay	Acc test	Acc control
0.5 sec	9.8% (±3.1%)	77.08% (±2.72%)
3 sec	7.9% (±0.8%)	77.08% (±2.72%)
5 sec	8.23% (±2.32%)	77.08% (±2.72%)

Table 5: Average accuracies and standard deviations (in parentheses) of classifier W for different delays of starting the background page load. The parameters for this experiment are: $n_{train} = 9$, $n_{test} = 1$, $T_{train} = 36$, $t_{test} = 4$, $m = 10$ and $k = 100$.

So far we showed the results obtained when the adversary is able to identify either the foreground page or the background page. We also consider the case where the user utilizes a countermeasure such as Camouflage [23]. In that case, the user is not interested in the contents of the background page thus the adversary is successful only if he is able to identify the foreground page. The accuracies obtained using this definition halved the accuracies showed in Table 5 (Acc test) and Figure 5.

4.6 Tor Browser Bundle Versions

In this section we evaluate the effect of different TBB versions and properties of TBB on WF. We evaluate the impact of having different TBB versions for training and testing. In practice, many TBB versions coexist, largely because of the lack of an auto-update functionality. It may be difficult for the attacker to know the exact version that is being used by the user. We also changed the configuration of Tor in the torrc to see how deviating from the default configuration may affect the success of the attack.

TBB Versions

We evaluate different combinations of TBB versions *2.4.7*, *3.5* and *3.5.2.1* for the control (training) and the test crawls. Table 6 shows the accuracy of classifier W when it trained on traces from Alexa Top 100 sites collected using TBB in the column and tested on the traces from the same sites collected using the TBB in the rows.

For versions 3.5 and 3.5.2.1 we observe high accuracies of W independently of the training and testing choices. This may imply that the countermeasure based on request randomization integrated in the TBB [24] may not be effective. On the other hand, when we evaluate 2.4.7 we observe low accuracies for combinations with 3.5. This is probably due to the major differences between the two versions. Version 3.5.2.1 is only a subversion of the TBB 3.5 which does not incorporate as many changes as the difference between 3.5 and 2.4.7.

TBB	2.4.7 (Test)	3.5 (Test)	3.5.2.1 (Test)
2.4.7 (Train)	62.70% (±2.8%)	29.93% (±2.54%)	12.30% (±1.47%)
3.5 (Train)	16.25% (±4.51%)	76.38% (±4.97%)	72.43% (±3.22%)
3.5.2.1 (Train)	6.51% (±1.15%)	66.75% (±3.68%)	79.58% (±2.45%)

Table 6: Entry in row X, column Y corresponds to the Acc Test (Step 2) and standard deviation (in parentheses) obtained by training in TBB version X and testing in TBB version Y. The configuration for these experiments is: $n_{train} = 9$, $n_{test} = 1$, $T_{train} = 36$, $t_{test} = 4$, $m = 10$ and $k = 100$.

TBB properties

We vary the following properties: UseEntryGuards and NumEntryGuards. UseEntryGuards indicates the policy for entry guard selection. It can take the following two values: *enabled*, Tor selects three entry guards for a long period of time; or *disabled*, picks one entry guard at random every time it builds a new circuit. NumEntryGuards sets the number of entry guards that will be available for the construction of a circuit (Default: 3). Note that even though we specify a value for these variables, we clean the Tor data directory after each batch crawl and, therefore, entry guards possibly change across batches.

We trained and tested on the control crawl for three different pairs of values (only Step 1), listed in Table 7. The default configuration is to choose an entry guard from a list of three possible entry guards (shown in the first row of Table 7). We also evaluated the setting used by Wang and Goldberg [32], which consists in disabling UseEntryGuards (second row in Table 7). Finally, we enabled UseEntry-

`Guards` but used a list of only one possible entry guard (third row in Table 7).

Entry guard config.	Acc control

`NumEntryGuards = 3`
`UseEntryGuards = 1`
	64.40% (±3.60%)

`UseEntryGuards` = 0	62.70% (±2.80%)
`NumEntryGuards` = 1 `UseEntryGuards` = 1	70.38% (±11.70%)

Table 7: Accuracy for different entry guard configurations. For these experiments we used the following parameters: $n_{train} = 9$, $n_{test} = 1$, $T_{train} = 36$, $t_{test} = 4$, $m = 10$ and $k = 100$.

In Table 7 we summarize the results for these three different configurations using classifier `W`. We can see that the standard deviation increases significantly for the case where we fix one entry guard. Even though we fix the entry guard for all circuits in a batch, since we remove the Tor data directory after each batch, we force the entry guard to change. On the other hand, allowing Tor to pick a different entry guard for each circuit results in a more balanced distribution because it is more likely that the same entry guards are being used in each single batch, thus there is lower variance across batches. We must clarify that these results are not concluding and there may be a different explanation for such difference in standard deviation.

4.7 Network

Another important variable that is being ruled out by the assumptions described in the previous section is the Internet connection. We suspect that it is unrealistic to assume the adversary is able to train using the same exact Internet connection as the user, especially in the non-targeted attack. For that, he might need more capabilities than the ones included in the basic model.

In this section we study the effect of different network locations on the accuracy of the `W` classifier. To that end, we crawled in three networks located in cities in different continents: Leuven, New York and Singapore.

Loc. Train	Loc. Test	Acc test	Acc control
Leuven	New York	8.83% (±2.87%)	66.95% (±2.87%)
Leuven	Singapore	9.33% (±0.98%)	66.95% (±2.87%)
Singapore	New York	68.53% (±3.24%)	76.40% (±5.99%)

Table 8: Accuracy for different network locations. The Acc test (Step 2) is calculated by training on data from *Location Train* and testing in data from *Location Test*. The parameters for the setting of these experiments are: $n_{train} = 9$, $n_{test} = 1$, $T_{train} = 36$, $t_{test} = 4$, $m = 10$ and $k = 100$.

Our results show that the accuracy drop between the crawls training on Leuven and testing in one of the other two locations is relatively greater than the accuracy drop observed in the experiments between Singapore and New York. Since the VM in Leuven is located within a university network and the other two VMs in data centers belonging to the same company, we attribute this difference to the fact that data center Internet connections tend to be closer to the Internet backbone. This could account for similar properties in the connection of the VMs in New York and Singapore that helped the classifier matching training and testing instances.

4.8 The importance of false positives

In this section we evaluate the open-world scenario in which an adversary monitors a small subset of the total number of pages that a user can visit, thus cannot train a classifier using every possible page.

Open-world

In the open-world scenario the adversary monitors a small number of pages and trains a classifier on traffic traces of both monitored and non-monitored pages. Following the approach described by Wang et al. [32], we assume the adversary monitors four pages: `google.com`, `facebook.com`, `wikipedia.org` and `twitter.com` and the rest of the pages in the Alexa Top 100 URLs are not monitored. We train a classifier using 36 traces for each of the Alexa Top 100 URLs, including the URLs of our monitored pages. To show the accuracy of the classifier in a true open-world scenario, we test it using four traces for each of the monitored sites plus one trace for each of the sites ranging from Alexa rank 151 to 34,710. For the classification, we assume the attacker is only interested in learning whether the user is visiting a monitored page or not, and not the exact URL that the user is visiting. did not train on.

The classifier `W` offers an accuracy over 90%, a true positive rate (TPR) of 80% that is almost constant, and the false positive rate (FPR) tends to 2.6% when we increase the size of the world (Figure 6).

The base rate fallacy

Prior WF studies used accuracy-based metrics to measure the success of the WF attack in the open-world. This approach however neglects the *base rate* or *prior*, that is the probability of a user visiting a monitored page a priori. As it has been pointed out recently within the Tor community [25], this constitutes a bias in the evaluation of the WF attack called the *base rate fallacy*. Despite reporting high accuracies and low FPR when the prior is low the success of the attack can be significantly lower.

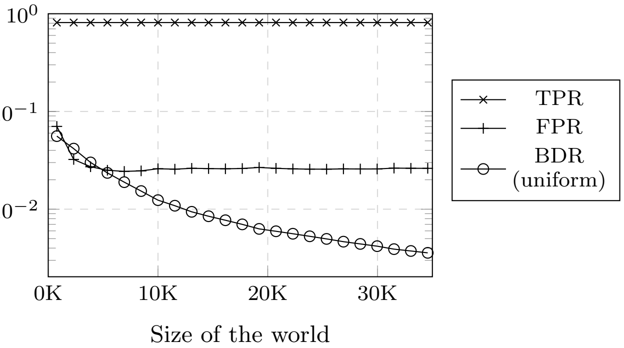


Figure 6: BDR in a uniformly distributed ∼35K open-world.

In contrast to prior work, we measure the success of the attack in the open-world using the *Bayesian detection rate* (BDR). The BDR is defined as the probability that a traffic trace actually corresponds to a monitored webpage given that the classifier recognized it as monitored.

Using the Bayes theorem, the BDR is expressed as

$$P(M \mid C) = \frac{P(C \mid M)\ P(M)}{P(M)\ P(C \mid M) + P(\neg M)\ P(C \mid \neg M)},$$

where M and C are the random variables of a webpage being monitored and a webpage being detected by the classifier as monitored respectively. We use the TPR as an approximation of $P(C \mid M)$ and the FPR to estimate $P(C \mid \neg M)$.

In Figure 6, we show the BDR with assuming a uniform distribution of web pages ($P(M) = \frac{|\text{Monitored}|}{|\text{World}|}$) along with the TPR and FPR of the classifier W for different sizes of the world. We observe that the BDR tends to zero as the size of the world increases. For a world of size 30K, which is a rather small world compared to the total size of the Web, the BDR is 0.4%. This means that there was a 0.4% probability that the classifier made a correct classification, and 99.6% of the times the adversary would wrongly conclude that the user was accessing a monitored page.

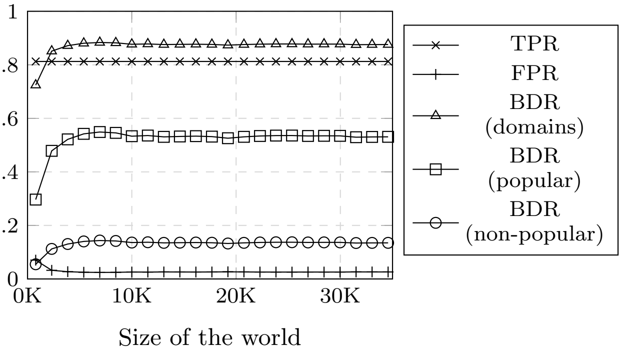


Figure 7: Evaluation of the BDR in a ∼35K open-world for multiple values of the prior $P(M)$. According to the ALAD data set, $P(M) = 0.18$ for 4 popular domains (*google.com*, *facebook.com*, *twitter.com* and *wikipedia.org*); $P(M) = 0.03$ for popular homepages and $P(M) = 0.005$ for non-popular homepages (4 random pages from ALAD with Alexa rank > 500).

Nevertheless, assuming a uniform distribution of pages introduces a statistical bias because it underestimates the probability of visiting popular pages. In order to give a more accurate estimation of the prior we extracted statistics about visits from the ALAD dataset. In particular, we measured frequency with which the users requested the URLs of the four monitored pages and its domains. We obtained $P(M) = 0.1852$ for domains and $P(M) = 0.0352$ for homepages. We found that in our case, where the attacker only identifies home pages, the BDR tends to just 53.1% (Figure 7).

We believe that a real-world adversary would not monitor homepages such as *google.com* or *facebook.com*. As a more plausible WF attack we have in mind a nation state-like adversary who is interested in identifying the access to specific pages that are difficult to block, such as an entry in a whistle-blower's blog hosted in a different country. These pages would presumably have a lower base rate than pages listed in Alexa.

To investigate the consequences of a very low prior we also calculated the BDR for monitoring a set of non-popular pages. We counted the number of visits in ALAD for 4 random pages that have a rank higher than 500 in Alexa, such as `careerbuilder.com` (rank 697) and `cracked.com` (rank 658). As expected, with a prior of 0.005 the BDR is much lower (marked with circles in Figure 7) and tending to 0.13%. Yet, note that these are just upper bounds because these monitored pages appear in the Alexa list and might be considered popular. We suspect that BDR for even more unpopular pages would be so low that would render a WF attack ineffective in this scenario.

User's browsing habits

In this section, we study the performance of a classifier that is trained on Alexa Top 100 sites and test it using real-world user visited sites. The goal is to check how successful an adversary, trained according to prior WF attacks, would be to find suspected pages on a set of pages a real-world user browsed.

We used the logs of three randomly chosen users from the ALAD and randomly picked 100 URLs from each. We crawled the URLs and collected data to feed the W classifier. During classification we mapped the test URLs to their top level domains. For example, `dell.msn.com` was mapped to `msn.com`. We chose to do this to not to overwhelm the classifier with false positives when it matches an inner page with the homepage. The results, summarized in Table 9 show a clear failure of the classifier in identifying the pages that these users were browsing.

ALAD User	TP	FP
User 3	38/260	362/400
User 13	56/356	344/400
User 42	3/208	397/400

Table 9: TPR and FPR for each of the users using a classifier trained on 36 traces from Alexa Top 100 sites and tested on randomly chosen 100 sites visited by ALAD User 3, 13 and 42. Here, $n_{train} = 4$, $n_{test} = 1$, $T_{train} = 36$, $t_{test} = 4$, $m = 10$ and $k = 100$.

Note that the true positive rate is the number of correct predictions over the number of predictions in which the page can be found in Alexa. The false positive rate is calculated as the number of misclassifications over the total number of predictions.

One possible reason for low TPR is due to the effect of *inner pages*. Inner pages are pages in the website that are not the homepage. We distinguish between two types of inner pages: (i) private, only accessible to the user through authentication (e.g., pages in Facebook or email accounts), and (ii), public, that is pages that are accessible for any web user but that it is not the homepage of the site. Other work has claimed that private inner pages do not matter because the TBB cleans session storage and a user has to load the login page after each session. However, we believe it is common that users leave the TBB open and visit the same inner page repeatedly within one single session. The high FPR is because the supervised classifier cannot output 'Unknown' for pages that do not exist in the training set, thus chooses to output a page in the training set that is the closest to the test page.

5. CLASSIFY-VERIFY

In standard supervised learning, a classifier chooses at least one class even when the target class is unavailable. For example, in the open-world scenario when a classifier is

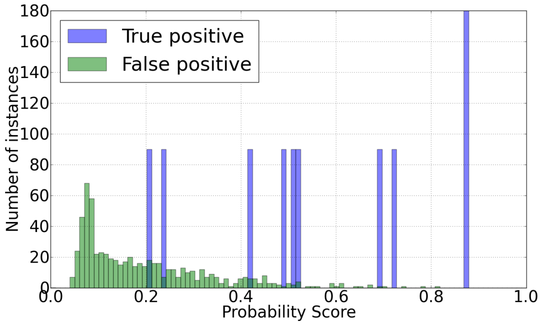


Figure 8: Estimated probability scores of the true positive and false positive instances during the open-world experiment (Section 4.8). Probability scores for the false positives are much lower than that of true positives. With a threshold ~0.2 most of the false positives can be discarded without reducing the true positive rate.

trained on web pages A, B and C and tested on page D, it will choose a page from A, B and C, although the original page (D) is absent. In the open-world experiment this introduces many false positives.

During classification, a classifier can output its confidence in the classification decision in terms of posterior probability. Although standard SVM classifier (classifier `W`) does not output probabilities, an additional sigmoid function can be trained to map the SVM outputs into probabilities[5]. One way to reduce the false positive rate is to inspect the probabilities estimated by the classifier and reject the classifier's decision when the probabilities are lower than a certain threshold. This is a form of Abstaining classification [5]. In this paper, we use the modified "Classify-Verify" approach as discussed by Stolerman et al. [27].

In the open-world experiment (Section 4.8), the probability scores for true positive instances are higher than the false positive instances (Figure 8). Note that, this was a multi-class classification with 100 classes, so the random probability score of an instance is 0.01.

The Classify-Verify approach adds an extra verification step after the classification. In our case, the verification process relies on a threshold that can be determined by training (Algorithm 1). When a page (D) is tested using the classifier, it outputs the probability scores of $D == A_i$ where $A_i \in \mathcal{A}$, sites in the training set. We use two verification scores based on these estimated probabilities: the maximum estimated probability, $P1$, and the difference between the maximum probability and the second highest probability, $Diff = P1 - P2$. If the verification score of D is less than the determined threshold, the classifier's output will be rejected. Unlike Stolerman et al. [27], we maximize F_β instead of $F1$ to choose threshold by adjusting weights for precision and recall. $\beta \leq 0.5$ achieves fewer false positives at the cost of true positives than $\beta > 0.5$. attacks.

Algorithm 1 Modified Classify-Verify

```
Input: Test page $D$, suspect pages $\mathcal{A} = A_1, ..A_n$ and probability scores
Output: $A_D$ if $A_D \in \mathcal{A}$ and 'Unknown' otherwise
  ▷ Train a classifier
  $C_\mathcal{A} \rightarrow$ classifier trained on $\mathcal{A}$
  $V_\mathcal{A} \rightarrow$ verifier for $\mathcal{A}$
  ▷ Calculate threshold for the verifier
  $t \rightarrow$ threshold maximizing $F_\beta$ score
  ▷ Test page D
  Classify D
  $P_D \rightarrow$ Verification score
  if $P_D >= t$ then
      Accept the classifier's output and return it
  else
      Reject the classifier's output and return 'Unknown'
  end if
```

5.1 Evaluation and result

We evaluate the Classify-Verify approach on the results of the open-world and ALAD experiments. To determine the threshold for a dataset, we use 10-fold cross-validation, where a threshold is determined by using 90% of the data and then tested on the remaining 10%. For F_β score, we choose $\beta = 0.5$ as we want to give more priority to precision than recall. We experimented with other β values and $F_{0.5}$ finds the best threshold (0.21 for open-world) that gives low false positives without reducing the TPR.

Our result shows that Classify-Verify reduces the number of false positives significantly without reducing the TPR. The new FPR after Classify-Verify is ~ 0.68. In the largest experiment (with $\sim$ 35K pages) the number of false positive reduces from 647 to 235, which is over 63% drop. The FPR can be reduced even further by sacrificing the TPR. The threshold estimated using $Diff = P1 - P2$ and $P1$ both perform similarly in our case.

Similarly for the users in ALAD, we determine the threshold using cross-validation. The number of false positive drops significantly (Table 10) over 50% for each of the three users.

ALAD User	TP	FP	New TP	New FP
User 3	38/260	362/400	31.2/260	107.6/400
User 13	56/356	344/400	26.8/356	32/400
User 42	3/208	397/400	1.0/208	41.2/400

Table 10: Classify-Verify result on the ALAD users. The number of FP drops by around 200.

The adversary can also use a pre-determined threshold instead of computing it every time. For example, in the open-world case if we had just chosen a threshold of 0.5 we could have discarded even more false positives with a little drop in true positives. Other more sophisticated approaches can be applied to choose a threshold, for example measuring the variance between intra- and inter-class instances. However, even after classify-verify the number false positive is lower than before but still very high. The most difficult cases are the high confidence false positives which indicate the cases where the features from censored and uncensored pages overlap.

We computed the BDR before and after applying the verification step. We used the estimation of the prior based on the prevalence of the homepages of the monitored websites in the ALAD dataset. The results show that the BDR doubles when we use the Classify-Verify approach. However, the BDR is still very low due to the exceptionally high FPR in this specific setting. For this reason, we conclude that

[5] In LibSVM, this can be achieved by simply using the `-b` option during training and testing [17].

Classify-Verify does not solve the issue completely but can be useful to partially mitigate the impact of false positives.

6. MODELING THE ADVERSARY'S COST

In this section, we model the cost of an adversary to maintain a WF system and discuss the scenarios in which the attack is most threatening. Current research considers only one scenario where the adversary has the maximum information about users. Even when the adversary has all possible information, collecting, maintaining and updating these information can be costly. For example, our anecdotal experience shows that the traffic footprint of Google homepage significantly changes due to different images (doodles) embedded on the page.

A typical WF system requires 4 tasks: data collection, training, testing and updating.

Data collection cost: At first the adversary needs to collect data from the training pages. In order to maximize the classifier accuracy, the adversary may want to train with different localized versions of the same webpage and collect these under different settings, e.g., different TBB versions, user settings, entry guard configurations. If we denote the number of training pages by n, and assume that on average webpages have m versions that are different enough to reduce the classifier's accuracy, the number of pages the adversary needs to collect is $D = n \times m \times i$, where i is the number of instances per page. We denote the data collection cost as $col(D)$. This cost includes both network and storage costs.

Training Cost: In the training phase an adversary needs to train his classifier with the collected data. The training cost includes the cost of measuring features F and training a classifier C. So the cost of training the system once would be $train(D, F, C)$. If c denotes the cost of training with a single instance of a traffic trace, then the cost of training the system once would be $train(D, F, C) = D \times c$.

Testing Cost: For testing a trace, the adversary needs to collect test data T, extract features F and test using the classifier C. Let v denote the number of monitored victims and p denote the average number of pages accessed by each victim per day. Then the amount of test data is $T = v \times p$. The total test cost is $col(T) + test(T, F, C)$.

Updating Cost: To maintain the performance of the classifier, the adversary needs to update the system over time. For example the adversary might try to keep the accuracy of the classifier above a certain threshold (e.g., 50%). The updating costs include the cost of updating the data (D), measuring the features (F) and retraining the classifier (C), which is denoted as $update(D, F, C)$. If, on average, webpages change d day periods, the daily updating cost would be $\frac{update(D,F,C)}{d}$.

Then, the total cost of an adversary to maintain a WF system is:

$$init(D, F, C, T) = col(D) + train(D, F, C) + col(T) + test(T, F, C)$$

$$cost(D, F, C, T) = init(D, F, C, T) + \frac{update(D, F, C)}{d}$$

To give a more concrete example, our experiment to measure the effect of time to classifier accuracy, we found that after $d = 10$ days the accuracy attained was under 50% and thus the adversary would have needed to update his data. The $\frac{update(D,F,C)}{10}$ would not show the impossibility of a successful WF attack, but it could show that to maintain such an attack can be prohibitively expensive even for adversaries with high level of resources.

The adversary could also have extra costs before the training and the data collection. For example, the attacker could try to discover background information about the victim(s) that can be used to increase the efficiency of the attack. He could also try to discover properties about the Internet connection of the user passively. However, in perspective of the results of the previous sections, the amount of background information required to mitigate the effect of noise in the data can be much larger than previously expected. Further, a question that lingers is to what extent, given such amount of background information, the WF attack is still necessary.

7. CONCLUSION AND FUTURE WORK

In this paper we studied the practical feasibility of WF attacks. We are not dismissing WF as a threat, but we suggest that more attention should be paid to the practicality of the scenarios in which these attacks are evaluated.

We studied a number of variables in isolation including website variance over time, multitab browsing behavior, TBB version, Internet connection, and the open world. When each of these assumptions are violated, the accuracy of the system drops significantly, and we have not examined in depth how the accuracy is impacted when multiple assumptions are violated.

Our results showed that success of a WF adversary depends on many factors such as temporal proximity of the training and testing traces, TBB versions used for training and testing, and users' browsing habits, which are commonly oversimplified in the WF models. Therefore, for most cases it seems that the non-targeted attack is not feasible given the sophistication level of current attacks.

There may be some exceptions in the case where a site of interest, perhaps a whistleblowing site, is particularly unique in its features and stable over time. Even the case of targeting a user is non-trivial, as these aspects of their behavior must be observed a priori or guessed correctly for WF to be a significant threat. Some users' behavior may be more susceptible to these attacks than others. In that case, the adversary could also have enough background knowledge to mount a more targeted attack and reduce the false positive rate that we demonstrated empirically to be critical for the success of the WF adversary.

We believe that further research on evaluating the common assumptions of the WF literature is important for assessing the practicality and the efficacy of the WF attacks. Future work in developing WF attacks against Tor should also evaluate their proposed attacks in practical scenarios, so that the Tor stakeholders and the research community have a more realistic assessment of the threat they are facing.

Acknowledgements

We thank our shepherd Rob Johnson and the anonymous reviewers for their feedback to improve this research. A special acknowledgement to Mike Perry for reviewing the draft version of the paper and debugging the Randomized Pipelining defense in Tor. We appreciate the interesting discussions with Kevin P. Dyer, Dominik Herrmann, Brad Miller, Andriy Panchenko, Tao Wang and Matthew Wright that helped clarifying this paper. This work was partially funded by the projects IWT SBO SPION, FWO G.0360.11N, FWO G.0686.11N, the KU Leuven BOF OT project ZKC6370 OT/13/070, NSF 1253418 and Intel ISTC for Secure Computing.

8. REFERENCES

[1] Alexa. Alexa Top 500 Global Site. `http://www.alexa.com/topsites`, 2014.

[2] G. D. Bissias and M. Liberatore. Privacy Vulnerabilities in Encrypted HTTP Streams. In *Privacy Enhancing Technologies (PETs)*, pages 1–11. Springer, 2006.

[3] X. Cai, X. Zhang, B. Joshi, and R. Johnson. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *ACM Conference on Computer and Communications Security (CCS)*, pages 605–616, 2012.

[4] H. Cheng and R. Avnur. Traffic Analysis of SSL Encrypted Web Browsing. *Project paper, University of Berkeley*, 1998. Available at `http://www.cs.berkeley.edu/~daw/teaching/cs261-f98/projects/final-reports/ronathan-heyning.ps`.

[5] C. Chow. On Optimum Recognition Error and Reject Tradeoff. *IEEE Transactions on Information Theory*, 16(1):41–46, 1970.

[6] S. E. Coull, M. P. Collins, C. V. Wright, F. Monrose, M. K. Reiter, et al. On Web Browsing Privacy in Anonymized NetFlows. In *USENIX Security Symposium*, pages 339–352. USENIX Association, 2007.

[7] G. Danezis. Traffic Analysis of the HTTP Protocol over TLS. *Unpublished draft*, 2009. Available at: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.3893&rep=rep1&type=pdf`.

[8] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*. USENIX Association, 2004.

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

[10] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Hiding Routing Information. In *Information Hiding*, pages 137–150. Springer, 1996.

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

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

[13] A. Houmansadr, C. Brubaker, and V. Shmatikov. The Parrot Is Dead: Observing Unobservable Network Communications. In *IEEE Symposium on Security and Privacy (S&P)*, pages 65–79. IEEE, 2013.

[14] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson. Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries. In *ACM Conference on Computer and Communications Security (CCS)*, pages 337–348. ACM, 2013.

[15] P. Juola, J. I. Noecker Jr, A. Stolerman, M. V. Ryan, P. Brennan, and R. Greenstadt. A Dataset for Active Linguistic Authentication. In *IFIP WG 11.9 International Conference on Digital Forensics.* Springer, 2013.

[16] M. Liberatore and B. N. Levine. Inferring the Source of Encrypted HTTP Connections. In *ACM Conference on Computer and Communications Security (CCS)*, pages 255–263. ACM, 2006.

[17] LibSVM. Multi-class classification (and probability output) via error-correcting codes. `\url{http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/#multi_class_classification_and_probability_output_via_error_correcting_codes.}`, 2014.

[18] L. Lu, E. Chang, and M. Chan. Website Fingerprinting and Identification Using Ordered Feature Sequences. In *European Symposium on Research in Computer Security (ESORICS)*, pages 199–214. Springer, 2010.

[19] X. Luo, P. Zhou, E. Chan, and W. Lee. HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. In *Network & Distributed System Security Symposium (NDSS)*, 2011.

[20] S. Mistry and B. Raman. Quantifying Traffic Analysis of Encrypted Web-Browsing. *Project paper, University of Berkeley*, 1998. Available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.10.5823\&rep=rep1\&type=pdf`.

[21] Mozilla Labs. Test Pilot: Tab Open/Close Study: Results. `https://testpilot.mozillalabs.com/testcases/tab-open-close/results.html#minmax`. (accessed: March 17, 2013).

[22] S. J. S. Murdoch and G. Danezis. Low-Cost Traffic Analysis of Tor. In *IEEE Symposium on Security and Privacy (S&P)*, pages 183–195. IEEE, 2005.

[23] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 103–114. ACM, 2011.

[24] M. Perry. Experimental Defense for Website Traffic Fingerprinting. Tor project Blog. `https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting`, 2011. (accessed: October 10, 2013).

[25] M. Perry. A Critique of Website Traffic Fingerprinting Attacks. Tor project Blog. `https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks`, 2013. (accessed: December 15, 2013).

[26] Y. Shi and K. Matsuura. Fingerprinting Attack on the Tor Anonymity System. In *Information and Communications Security*, pages 425–438. Springer, 2009.

[27] A. Stolerman, R. Overdorf, S. Afroz, and R. Greenstadt. Classify, but verify: Breaking the closed-world assumption in stylometric authorship attribution. In *IFIP Working Group 11.9 on Digital Forensics.* IFIP, 2014.

[28] Q. Sun, D. R. Simon, and Y. M. Wang. Statistical Identification of Encrypted Web Browsing Traffic. In *IEEE Symposium on Security and Privacy (S&P)*, pages 19–30. IEEE, 2002.

[29] Tor project. Users statistics. `https://metrics.torproject.org/users.html`. (accessed: March 20, 2013).

[30] Tor project. Welcome to Stem! Stem 1.1.1 Documentation. `https://stem.torproject.org`, 2014.

[31] C. von der Weth and M. Hauswirth. DOBBS: Towards a Comprehensive Dataset to Study the Browsing Behavior of Online Users. *CoRR*, abs/1307.1542, 2013.

[32] T. Wang and I. Goldberg. Improved Website Fingerprinting on Tor. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 201–212. ACM, 2013.

[33] T. Wang and I. Goldberg. Comparing Website Fingerprinting Attacks and Defenses. 2014. Technical report.

APPENDIX

A. LIST OF USED CRAWLS

Crawl Name	**Date**	**Network**	**Version**	**Size**	**Batches**	**Accuracy control**	**Std**
140203_042843	2/3/2014	Leuven	3.5	100	10	77.08%	± 2.72%
140203_040706	2/3/2014	Leuven	2.4.7 Alpha 1	100	10	62.70%	± 2.80%
140209_162439	2/9/2014	New York	2.4.7 Alpha 1	100	10	67.53%	± 3.91%
140214_050040	2/14/2014	Singapore	3.5	100	10	78.70%	± 4.01%
140220_042351	2/20/2014	New York	2.4.7 Alpha 1	100	10	66.05%	± 3.42%
140325_115501	3/25/2014	New York	3.5.2.1	100	10	79.58%	± 2.45%
140329_194121	3/29/2014	Singapore	3.5.2.1	100	10	76.40%	± 5.99%
140329_191630	3/29/2014	Leuven	3.5	100	10	66.95%	± 2.87%
140418_145104	4/18/2014	Leuven	3.5	100	6	54.46%	± 21.15%
140426_021609	4/26/2014	Singapore	3.5	100	10	76.93%	± 3.86%
140427_140222	4/27/2014	Leuven	3.5	100	10	71.35%	± 9.09%
140506_224307	5/7/2014	New York	3.5	100	10	77.05%	± 6.29%
140508_144031	5/8/2014	New York	3.5	100	10	72.73%	± 3.18%
140329_184252	3/29/2014	Leuven	3.5	100	10	70.38%	± 11.72%
140210_201439	2/10/2014	Leuven	2.4.7 Alpha 1	100	10	66.88%	± 5.16%
140214_040009	2/14/2014	Leuven	3.5	100	5	64.40%	± 3.60%