



# Touching from a Distance: Website Fingerprinting Attacks and Defenses

Xiang Cai  
Stony Brook University  
xcai@cs.stonybrook.edu

Xin Cheng Zhang  
Stony Brook University  
xinczhan@gmail.com

Brijesh Joshi  
Stony Brook University  
sunjosh17@hotmail.com

Rob Johnson  
Stony Brook University  
rob@cs.stonybrook.edu

## ABSTRACT

We present a novel web page fingerprinting attack that is able to defeat several recently proposed defenses against traffic analysis attacks, including the application-level defenses HTTPPOS [15] and randomized pipelining over Tor [18]. Regardless of the defense scheme, our attack was able to guess which of 100 web pages a victim was visiting at least 50% of the time and, with some defenses, over 90% of the time. Our attack is based on a simple model of network behavior and out-performs previously proposed ad hoc attacks. We then build a web site fingerprinting attack that is able to identify whether a victim is visiting a particular web site with over 90% accuracy in our experiments.

Our results strongly suggest that ad hoc defenses against traffic analysis are not likely to succeed. Consequently, we describe a defense scheme that provides provable security properties, albeit with potentially higher overheads.

## Categories and Subject Descriptors

C.2.0 [Computer-Communication Networks]: General—*Security and protection*

## Keywords

Anonymity, website fingerprinting attacks

## 1. INTRODUCTION

Web browsing privacy mechanisms, such as SSL, Tor, and encrypting tunnels, hide the content of the data transferred, but they do not obscure the size, direction, and timing of packets transmitted between clients and remote servers. In a web page fingerprinting attack, an adversary attempts to use this information to identify the web page a victim is visiting. Previous research has shown that web page fingerprinting attacks are possible against many privacy services, including IPsec tunnels, SSH tunnels, and Tor [21, 10, 17, 6, 13].

As a result, researchers have proposed several defenses, primarily aimed at hiding packet size information. For ex-

ample, Tor packs all data into 512-byte cells. Other mechanisms pad packets in a variety of ways (e.g. padding to  $2^k$  bytes, or padding all packets to the MTU). Wright, et al., proposed traffic morphing, which pads and fragments packets so that the resulting distribution of packet sizes appears to be from a different web page [26]. Dyer, et al. showed that all these schemes are broken [6].

CCS'12, October 16–18, 2012, Raleigh, North Carolina, USA.  
Copyright 2012 ACM 978-1-4503-1651-4/12/10 ...\$10.00.

Defense	Rate
None (SSH tunnel)	91.6%
SSH + HTTPPOS	75.7%
SSH + Sample-based morphing	92.1%
Tor	83.7%
Tor + randomized pipelining	87.3%
Tor + rand. pipe. + rand. traffic	52.2%

Table 1: Success rate of our web page fingerprinting attack against each defense evaluated in this paper. The success rate is the probability that the attack was able to correctly guess which of 100 web pages the victim was visiting.

Researchers have recently proposed defenses based on manipulating the sequence and structure of the HTTP requests generated by the browser. HTTPPOS, published at NDSS 2011, manipulates TCP MSS and window size parameters to obscure packet sizes, but also includes several HTTP-specific mechanisms [15]. For example, HTTPPOS can split individual HTTP requests into multiple partial requests, can issue extra HTTP requests as cover traffic, and can use pipelining to execute requests concurrently, obscuring the exact order of requests. Pipelining, which was originally introduced to improve performance, allows web clients to issue subsequent requests without waiting for the response from previous requests. Similarly to HTTPPOS, the Tor project has released a version of Firefox that implements “randomized pipelining,” in which the browser requests objects in a random order and with random levels of pipelining [18].

In this paper, we demonstrate effective attacks against HTTPPOS, randomized pipelining, and several other defenses. Table 1 summarizes the results of our attack on each of defense we evaluate. Our attack can determine, with a success rate over 83%, which of 100 web page a victim is visiting via Tor, even if the victim uses randomized pipelining. Against SSH tunnels, our attack could determine which web page the victim was visiting over 75% of the time, even if the victim used HTTPPOS or sample-based traffic morphing. We also evaluated our attack against a simulated Tor implementation that used randomized pipelining, padded all packets to

Attack	Defense	Database Size	Success Rate
Our page fingerprinting attack	Tor	100	<b>83.7%</b>
Ad hoc SVM [17]	Tor	100	65.4%
Cosine Similarity [19]	Tor	20	50%
Multinomial Naive-Bayes [10]	Tor	100	4.4%
Our page fingerprinting attack	Tor + rand. pipe.	100	<b>87.3%</b>
Ad hoc SVM [17]	Tor + rand. pipe.	100	62.8%
Our page fingerprinting attack	None (SSH)	100	91.6%
Ad hoc SVM [17]	None (SSH)	100	<b>92.0%</b>
Multinomial Naive-Bayes [10]	None (SSH)	100	81.9%

**Table 2: The success rates of our attacks compared to relevant previous attacks. The results for the cosine similarity classifier are taken from Shi, et al [19]. All other results are computed using our implementations on our data sets.**

1500 bytes, and added random cover traffic. Even with a 1-to-1 ratio between cover traffic and real traffic, our attack could identify the victim’s web page over 50% of the time.

Ours is the first demonstration that application-level defenses, such as HTTPS and randomized pipelining, are not secure. All previous attacks have only shown that defenses based solely on packet padding and similar network-level manipulations were not effective. We also compare our attack to several previously published attacks, as shown in Table 2. In 2009, Herrmann, et al., proposed a fingerprinting attack based on a Multinomial Naive-Bayes classifier [10], which, in our experiments is able to identify which web page a victim visited (out of a set of 100 possible pages) with a success rate of less than 5%. Our attack has over an 80% success rate under similar conditions. Shi, et al., proposed a fingerprinting attack based on cosine similarity in 2009 [19], but this method had a success rate of only 50%, even when there were only 20 web pages to choose from. In 2011, Panchenko, et al. published a classifier using ad hoc HTTP-specific features, but it only achieves a 65% success rate on our data set [17]. Our attack also works well against simple SSH-tunneled traffic, achieving a 92% success rate, comparable to the rate achieved by Panchenko et al.’s classifier and the VNG++ classifier of Dyer et al. [6].

Our attack has two novel components. First, we propose a new method for computing the similarity of packet traces generated when a browser loads a web page. Our attack converts traces into strings and uses the Damerau-Levenshtein distance to compare them. Packet ordering is useful for identifying web pages because the order of incoming and outgoing packets reveals information about the size of objects referenced in a page and the order in which the browser requests them. Damerau-Levenshtein distance is a good metric because it allows insertions, deletions, substitutions, and transpositions, operations that correspond well with network packet drops, retransmissions, and re-orderings, and with slight changes in a page’s content, as may occur with pages dynamically generated from a template.

We then use Hidden Markov Models to extend our web *page* classifier to a web *site* classifier. An attacker can use these models to determine if a sequence of a victim’s page loads are all from the same web site. The HMM captures the link structure of the site and the probable paths that users will follow among the pages when visiting the site. The HMM uses our novel page fingerprinting technique to classify the packet traces observed each time the user transitions from one page to another. The attacker can then use

the Forward algorithm to compute the probability that an observed trace of packets was generated by a browser loading pages from the target web site. Our site classifier was able to identify when a user visited a target web site via Tor with over 90% accuracy in our experiments.

Our results strongly suggest that the ad hoc approach to traffic analysis defenses taken so far, in which researchers design defenses in response to new attacks, will not lead to secure systems. We advocate that researchers adopt a provable security approach to traffic analysis defense design.

As a first step in this direction, we present an extension of the BUFL0 scheme proposed by Dyer et al. [6]. Our scheme addresses practical, performance, and security shortcomings of the original protocol. Our defense offers provable security properties, but may incur higher bandwidth or latency overhead than previously-proposed defenses.

This paper makes the following contributions:

- We show that recently proposed application-level defenses, such as HTTPS and randomized pipelining, are not secure.
- We present a new web page fingerprinting attack that significantly outperforms other proposed attacks on these and other defenses. Our attack can determine which web page, out of 100 possibilities, a victim is visiting with over 80% success rate.
- We present a novel web site fingerprinting attack that can identify, with over 90% accuracy, when a victim is visiting a particular web site.
- We propose a traffic analysis defense with provable security properties.

## 2. RELATED WORK

Researchers have studied attacks on anonymity systems from a variety of angles: active attacks that require subverting nodes in the anonymity network, active attacks that require injecting traffic into the network, and attacks based on subverting web servers visited by anonymous users. Some of these attacks focus on discovering the identity of the anonymous network user, others focus on discovering the servers they interact with, and others attempt to uncover the victim’s route through the anonymizing network.

Web page fingerprinting attacks are an important class of attacks because they are a good match for the attacker scenario faced by many Tor users today: they use Tor to

evasion censorship and persecution by a government or ISP that wants to know their browsing habits and has the ability to monitor their internet connection, but cannot easily infiltrate Tor nodes and web servers outside the country.

**Fingerprinting attacks on encrypting tunnels.** Several researchers have developed web page fingerprinting attacks on encrypted web traffic, as occurs when the victim uses HTTPS, link-level encryption, such as WPA, or an encrypting tunnel such as SSH, a VPN, or IPSec [2, 4, 9, 10, 11, 13, 14, 20, 27, 28, 6]. Most attacks against these systems focus on packet sizes, and many throw away all information about packet ordering. Packet sizes do carry a lot of information in these scenarios, where data packets are simply padded to a multiple of the block size (typically 16 bytes), but Tor pads all data packets to a multiple of 512 bytes, providing much less information. Most recently, Dyer et al. performed a thorough survey of past attacks and past network-level defenses and found that no network-level defense was secure [6]. They did not evaluate application-level defenses, such as HTTPPOS or randomized pipelining.

The unpublished work of Danezis [4] is also worth pointing out, since it uses HMMs to model entire web sites in much the same way that we do. Lu, et al., propose a fingerprint based on edit distance [14], but their fingerprints depend heavily on packet size information, which is not available when attacking Tor users. Yu, et al. [27] also proposed to use HMMs to model web sites, but their observations consisted only of the amount of time a victim spent viewing each page, and hence their success rate was not very high.

**Fingerprinting attacks on Tor.** There is relatively little research on fingerprinting attacks on Tor. Herrmann, et al., used a Multinomial Naive Bayes classifier on features that captured no information about packet ordering – only packet sizes [10]. They applied this classifier to several encrypting tunnels, such as SSH, and achieved over 94% success in recognizing packet traces from a set of 775 possible web pages. When they applied this classifier to Tor, however, they had less than a 3% success rate on the same set of web pages. In the same year, Shi, et al., proposed to use cosine similarity on feature vectors that represented some ordering information about packets, but they achieved only a 50% success rate on a set of 20 web pages [19]. Panchenko, et al., used ad hoc, HTTP-specific features with support vector machines to achieve a 54.61% success rate on the same data set [17]. We re-implemented their attack and obtained a 65.4% success rate on our data set of 100 web pages.

**Proposed traffic analysis defenses.** IP- and TCP-level defense mechanisms involve padding packets, splitting packets into multiple packets, and inserting dummy packets. Fu, et al., performed an early theoretical analysis of constant-rate transmission of fixed-size packets as a defense mechanism [8]. Surprisingly, they found that variations in load at the sender caused detectable variations in transmission time, implying that transmitting at random intervals provides better defense against analysis. Wright, et al., proposed a technique for morphing one traffic pattern to look like another pattern [26]. Their morphing algorithm only mapped one packet size distribution onto another – it did not change the sequencing of packets or handle correlations between the sizes of successive packets. They also proposed a variant of their defense that would only enlarge packets – it never split or re-ordered packets. Since our attack works well even without packet size information, it can defeat this ver-

sion of traffic morphing (our experiments achieved an 81% success rate, described later). Lu, et al., later analyzed traffic morphing, including an extension to morphing on the distribution of packet size n-grams [14].

At NDSS 2011, Luo, et al., described HTTPPOS, a collection of HTTP- and TCP-level tricks for fooling traffic analysis attacks previously described in the literature [15]. At the TCP level, they manipulate MSS options and window sizes to perturb the size and ordering of packets in the TCP stream. At the HTTP level, they split single requests into multiple possibly overlapping requests using the HTTP Range feature, re-order some requests via pipelining, generate some extra, unnecessary requests, and insert some extra data into HTTP GET headers. Our attack is able to defeat their prototype implementation of HTTPPOS.

The Tor project recently proposed a traffic analysis defense based on “randomized pipelining”, in which the browser loads images and other embedded content in a random order [18]. It also pipelines random subsets of these requests. Even with this defense in place, our attack is able to identify the target web page over 87% of the time in our experiments.

**Other related work.** A few previous papers are notable for using similar techniques on similar problems. Wright, et al., used HMMs for protocol classification of encrypted TCP streams [25], i.e. to determine whether an encrypted connection was an HTTP, SMTP, POP, IMAP, etc. session. More recently, White, et al., used HMMs to recover partial plaintext of encrypted VoIP conversations [24].

### 3. RECOGNIZING WEB PAGES

Web pages can consist of multiple objects, such as HTML files, images, and flash objects, and browsers send separate requests for each object. Browsers may use a combination of multiple TCP connections and pipelining in order to load pages more quickly [12]. Furthermore, browsers may begin issuing requests for objects referenced in a web page before they have finished loading that page.

Note, however, that there is some inherent stability in the ordering of requests: browsers cannot request an object until they have received the portion of a page that references it. The sequence of requests and responses may vary each time the browser loads the page: some requests may be delayed due to CPU load or packet re-ordering, and some requests (or responses) may be omitted if the browser has a copy of the object in its cache. Dynamic web pages may also vary slightly in the size and number of objects they contain, and hence in the number of requests sent by the browser and the total number of packets returned by the server.

Web privacy proxies, such as Tor and SSH, multiplex these data transfers over a single, encrypted channel, so an attacker can only see the size, direction, and timing of packets in the multiplexed stream. Tor furthermore sends all data in 512-byte cells, so packet sizes carry limited information.

These facts suggest a simple representation for the attacker’s traffic observations, and a similarity metric the attacker can use to compare traces. Our attack represents a trace of  $\ell$  packets as a vector  $t = (d_1, \dots, d_\ell)$ , where  $d_i = \pm s_i$ , where  $s_i$  is the size of the  $i$ th packet and the sign indicates the direction of the packet. Our attack compares traces  $t$  and  $t'$  using the Damerau-Levenshtein edit distance [16], which is the length of the shortest sequence of character insertions, deletions, substitutions, and transpositions required to transform  $t$  into  $t'$ . In the context of

our packet traces, these edits correspond to packet and request re-ordering, request omissions (e.g. due to caching), and slight variations in the sizes of requests and responses. Thus, this model and distance metric are a good match for real network and HTTP-level behavior.

The Damerau-Levenshtein algorithm supports different costs for each operation. Ideally, these costs would be tuned to match the probability of packet drops, retransmissions, etc. in the real network. We experimented with several cost schemes; the impact was mild, but the attack yielded best results when transpositions were 20 times cheaper than insertions, deletions, and substitutions. We did not explore this parameter thoroughly – a better approach would be to learn optimal costs from the training data using the recently-proposed method of Bellet, et al. [1].

We found that TCP ACK packets reduce the performance of our classifier. This seems natural: inserting an ACK after every packet essentially makes all traces look more similar – they’re all half ACKs. Our Tor classifier deletes all 40 and 52 byte packets from the traces. Our SSH classifier deletes all packets of size 84 or less.

Since Tor transmits data in 512-byte cells, our attack also rounds all packet sizes up to a multiple of 600 (we use 600 instead of 512 in order to account for other inter-cell headers and overhead). In some of the experiments described in Section 6, we deleted all packet size information, i.e. traces were reduced to sequences of  $\pm 1$ s.

Our attack normalizes the edit distance to compensate for the large variation in the lengths of packet traces. If  $d(t, t')$  is the Damerau-Levenshtein edit distance, the attack uses

$$L(t, t') = \frac{d(t, t')}{\min(|t|, |t'|)}$$

where  $|t|$  is the number of packets in trace  $t$ . The classifier normalizes by the minimum of the two lengths because, if  $t$  and  $t'$  are very different in length, then they are probably from different web pages. In this case, dividing by  $\min(|t|, |t'|)$  will result in a relatively large normalized distance, which is desirable. Other normalization factors, such as  $|t| + |t'|$  and  $\max(|t|, |t'|)$ , yielded worse results.

To build a classifier for recognizing encrypted, anonymized page loads of 1 of  $n$  web pages, an attacker collects  $k$  traces of each page, using the same privacy system, e.g. Tor or an SSH proxy, in use by the victim. He then trains a support vector machine [22] using a kernel based on edit distance:

$$K(t, t') = \exp(-\gamma L(t, t')^2)$$

The  $\gamma$  parameter is used to normalize  $L$  so that it’s outputs fall into a useful range. In our experiments, we found  $\gamma = 1$  works well. We also adjusted the SVM cost of misclassifications to be 4, based on early experimental results.

Intuitively, an SVM kernel function acts as an inner product on a vector space, allowing the SVM to measure the angle between two vectors. Vectors with a small angle are considered more similar by the SVM and likely to be placed in the same class. The above kernel will assign traces with a small distance an “inner product” close to 1, indicating a small angle between them and hence high similarity. Traces with a large distance will have kernel value close to 0, corresponding to a large angle and hence low similarity.

This basic approach can be customized in several ways, depending on the application. For example, instead of viewing the observed network traffic as a sequence of packets, as

above, an attacker could view it as a sequence of 512-byte Tor cells, or even as a sequence of bytes, if appropriate. He would then generate a trace vector of  $\pm 1$ s for each cell or byte of traffic. Finally, the attacker could encode timing information by inserting additional “pause” symbols into the trace whenever there is a long gap between packets.

We briefly explored several of the above variations in our attack on Tor. We tried representing traces as a sequence of Tor cells instead of as a sequence of packets. Classifier performance degraded slightly, suggesting that the Tor cells are often grouped into packets in the same way each time a page is loaded. We tried adding pause symbols to our traces, but this made no contribution to classifier performance. An early version of our attack classified traces using a nearest neighbor algorithm: to classify trace  $t$ , the attacker computed  $t^* = \operatorname{argmin}_{t'} L(t, t')$  over every trace in his database, and guessed that  $t$  was from the same web page as  $t^*$ . This attack correctly guessed a victim’s web page (out of 100 possibilities) over 60% of the time. Finally, we tried using a metric embedding to convert our variable-length trace vectors into fixed-length vectors in a space using the  $\ell^2$ -norm, and then used an SVM to classify these vectors. This performed substantially worse than the SVM classifier with distance-based kernel described above.

## 4. RECOGNIZING WEB SITES

As the evaluation results in Section 6 will show, the classifier described above is quite good at determining which of  $n$  web pages a user is visiting, assuming the user is visiting one of those  $n$  pages. However, attackers often want to answer a slightly different question: “Is the user visiting one of a small list of banned web sites?” There are three differences between the previous scenario and this one: (1) there is no prior assumption about which sites the user may be visiting; (2) the attacker wants to know if the user is visiting any of the pages on a banned web site; and (3) the attacker will want a high degree of confidence in the answer.

To answer this type of question, an attacker can construct a Hidden Markov Model for each target web site, and use the forward algorithm to compute the log-likelihood that a given packet trace would be generated by a user visiting the target web site. If the log-likelihood is below a certain threshold, then he can conclude that the user is visiting the web site, otherwise she is not.

In our web site model, each web page corresponds to an HMM state, and state transition probabilities represent the probability that a user would navigate from one page to another. These transition probabilities, along with the initial state probabilities, can be derived from the link structure of the web site and observations of real user behavior.

To complete the HMM, the attacker must define the set,  $O$ , of observations and, for each observation  $o \in O$  and HMM state  $s$ , the probability,  $\Pr[o|s]$ , that the HMM generates observation  $o$  upon transitioning to state  $s$ . Our attack uses the classifier from the previous section for this purpose. The attacker collects  $k$  traces of each page in the target web site, along with  $k$  traces of  $n$  other web pages chosen arbitrarily (e.g. random web pages). These web pages form  $O$ , the set of observations that may be generated by the HMM. He uses the collected traces to build a classifier,  $C$ , as described in the previous section. For each page,  $s$ , in the target web site, he then collects  $\ell$  additional traces and estimates  $\Pr[o|s]$  as the fraction of the  $\ell$  traces from page  $s$  that  $C$  classifies as

page  $o$ . If no trace for a page  $s$  ever gets classified as a trace for page  $o$ , then he sets  $\Pr[o|s]$  to a small non-zero value.

Huge web sites may have thousands or even millions of pages, so it would be impractical to make a model covering each page separately. Fortunately, most large sites have pages that are constructed from templates. For example, Amazon.com has page templates for search results, individual items, reviews, etc. To handle large web sites, an attacker can create a model with states corresponding to page templates rather than individual pages. A set of web pages can be modeled as a single HMM state only if all the pages produce similar probability distributions of observations. In other words, pages  $p_1$  and  $p_2$  can be represented by a single state  $s$  only if  $\Pr[o|p_1] \approx \Pr[o|p_2]$  for all observations  $o$ . Experimental results in Section 6 will show that this is the case for pages generated from the same template.

HMM web site models can also handle pages that use AJAX. If a page can make  $r$  different requests to a web server, then the HMM can represent the page with  $r + 1$  states  $s_0, \dots, s_r$ . State  $s_0$  corresponds to the initial page load, and states  $s_1, \dots, s_r$  correspond to each AJAX transaction the page may execute. The attacker then treats AJAX operations like any other page load: he collects traces of the transactions, adds them to the classifier described above, and uses them to compute a probability distribution on observations. Other pages can only transition to  $s_0$ , but the transitions among states  $s_0, \dots, s_r$ , and transitions from the  $s_i$ s to other pages, are determined by the structure of the AJAX code. The probability of these transitions is determined by the code and by user behavior.

As a user traverses the pages of a web site, his browser collects a cache of page elements it encounters. The attacker must account for the browser cache when constructing an HMM for the site. Cold pages are unlikely to have elements cached in the browser. For example, a login page is typically visited once at the beginning of a session, and hence is “cold”. Warm pages may be loaded repeatedly or after the browser has collected a large cache. A user’s Facebook profile page is likely to be “warm”. An attacker can include both types of page in his model. For example, when modeling a social networking site, an attacker could model the login page as cold, and he could include both a cold and warm version of a user’s main profile page. The model would initially transition to the cold version of the profile page, but transitions from other states would go to the warm version.

Users may also move between pages using their browser’s “Back” and “Forward” buttons and by typing a URL directly into the location bar. The attacker can model page loads via the location bar by simply adding edges between states of the HMM. The probability assigned to these transitions can be derived from user behavior. Unfortunately, it is not possible to precisely model the Back and Forward buttons using an HMM, since that would require augmenting the HMM with a stack. In most browsers, clicking the Back button generates the same traffic trace as clicking a link to the previous page, so the attacker can model the Back button by adding reverse edges for every edge in the original HMM. Note that, since clicking back necessarily is a “warm cache” load of the previous page, the HMM back edge should go to the HMM state representing a warm cache load of the page, even if its corresponding forward edge is from a cold cache state. The probability assigned to each back edge can be derived from observing real users.

Note that this HMM-based attack assumes that users all tend to navigate through a website in the same way. If this assumption is not valid, e.g. if users have wildly differing habits when visiting the target site, then the attacker has two options. First, if user’s tend to follow one of a small set of different patterns, then the attacker can build an HMM for each pattern. If each user tends to have a totally unique pattern, then the attacker can assign uniform transition probabilities. The HMM will not use any ordering information, but it will still be able to make classification decisions based on the set of pages visited by the victim.

## 5. Congestion-Sensitive BUFL0

We now develop a traffic analysis defense with provable security properties. Our defense builds on the simple BUFL0 scheme defined by Dyer, et al. [6], but solves several practical, performance, and security problems of that scheme. We are currently working to implement and evaluate the Congestion-Sensitive BUFL0 algorithm, so we provide only a rough analysis of its performance and security below.

A  $(d, \rho, \tau)$  BUFL0 implementation transmits  $d$ -byte packets every  $\rho$  milliseconds, and continues this process for at least  $\tau$  milliseconds. If  $d$  bytes of application data are not available when a packet is to be transmitted, then BUFL0 fills any extra space, possibly the entire packet, with junk data that will be discarded at the other end. BUFL0 assumes the application can signal the beginning and end of its communications. If, after  $\tau$  milliseconds, the application has not completed its transmissions, then BUFL0 continues transmitting  $d$ -byte packets every  $\rho$  milliseconds until the application signals that it is finished.

The basic BUFL0 protocol has three shortcomings:

- High overhead. Depending on the configuration parameters, Dyer, et al. found that BUFL0 has an average bandwidth overhead between 93% and 419%. Configurations with lower overhead offered much less protection against the attacks surveyed in their paper.
- Low practicality. BUFL0 has no provisions for responding to congestion or flow control signals.
- Unclear security. When the application takes longer than  $\tau$  milliseconds to finish, BUFL0 reveals some information about the amount of data being communicated. As a result, in some BUFL0 configurations they evaluated, an attacker could guess the victim’s target web page (out of 128 pages) over 24% of the time.

The Congestion-Sensitive BUFL0 algorithm, shown in Figure 1, tunes its inter-packet transmission time,  $T$ , based on the data source. The algorithm operates on an input queue and an output queue. Data from the application arrives and is placed into the input queue. Data in the output queue is transmitted using a congestion- and flow-control aware protocol, such as TCP. Congestion-Sensitive BUFL0 monitors the output queue every  $T$  milliseconds and enqueues new data only when the output queue contains fewer than  $S$  cells. If the network becomes congested, then the sender process will stop transmitting (and removing) elements from the output queue. When the output queue grows to size  $S$ , then Congestion-Sensitive BUFL0 stops enqueueing more items until the transmission process is able to successfully transmit more cells (and remove them

```

procedure SCBUFLO(srcID)
  T = LOOKUP-SPEED(srcID)
  ncells = 0
  while SENDER-ACTIVE() OR !IS-EMPTY(input-queue) OR
    !IS-POWER-OF-TWO(ncells)
    if SIZE(output-queue) < S
      if IS-EMPTY(input-queue)
        ENQUEUE(output-queue, JUNK-CELL())
      else
        ENQUEUE(output-queue, DEQUEUE(input-queue))
    ncells = ncells + 1
    SLEEP(T)

```

**Figure 1:** Pseudo-code for the basic Congestion-Sensitive BUFL0 algorithm. For simplicity, this version assumes fixed-sized cells.

from the output queue). This algorithm still hides all information about the timing of incoming cells, though, since the sequence of cells enqueued in the output queue is independent of the arrival of cells in the input queue.

The parameter  $T$  governs the maximum transmission rate of the Congestion-Sensitive BUFL0 algorithm. The algorithm will transmit at most  $1000/T$  cells per second, but may transmit less if the outbound connection has a lower bottleneck bandwidth. Therefore, one may view Congestion-Sensitive BUFL0 as a link, with bandwidth  $1000/T$ , in the overall network path between the sender and receiver. In order to have good performance, Congestion-Sensitive BUFL0 should not be the bottleneck link, so  $1000/T$  should be large, i.e.  $T$  should be small. On the other hand, in order to avoid sending too many junk cells,  $T$  should be large.

We would ideally set  $T$  equal to the incoming packet inter-arrival time. Thus, Congestion-Sensitive BUFL0 would neither be the bottleneck link nor would it need to send a large number of junk cells. The algorithm in Figure 1 selects  $T$  using a database of data sources. In the context of web browsing, a source ID could be the URL of the page being loaded or simply the domain name of the server providing the page. The database mapping IDs to  $T$  values would be updated periodically based on recent measurements.

Tuning  $T$  to the source of the incoming data obviously may reveal some information about the data source to an attacker observing the outbound data link. Therefore, we must quantize the possible values of  $T$ . One simple choice would be to limit  $T$  to values of the form  $2^i$ , where  $i \in \mathbb{Z}$ .

The only other side information revealed to the attacker is,  $B$ , the number of transmitted cells, which Congestion-Sensitive BUFL0 quantizes to a power of 2. Although this may in the worst case double the amount of data transmitted, it can on average have a much lower overhead. Let  $x$  be the number of cells that would be transmitted if Congestion-Sensitive BUFL0 stopped transmitting as soon as the sender became inactive and the input queue was empty. If, for real data sources,  $x$  is uniformly distributed between  $2^{\lfloor \log_2(x) \rfloor}$  and  $2^{\lfloor \log_2(x) \rfloor + 1}$ , then the average overhead of padding the total transmission to  $2^{\lfloor \log_2(x) \rfloor + 1}$  cells is  $\int_1^2 \frac{2}{x} dx < 1.39$ .

In summary, we can control the overhead of the Congestion-Sensitive BUFL0 algorithm by tuning  $T$  to the website being loaded, and padding all transmissions to a power of 2 cells will add an additional overhead of only 40%, which, as

our evaluation in Section 6 will show, is competitive with the overheads of many of the schemes defeated in this paper.

We've presented Congestion-Sensitive BUFL0 as a unidirectional protocol. For web applications, each side will run an instance of the Congestion-Sensitive BUFL0 protocol. Each instance will reveal two pieces of side-information to an attacker:  $T$  and  $B$ . Thus, in total, the attacker is able to observe only the  $O = (T_{up}, T_{down}, B_{up}, B_{down})$ , where each of these values has been quantized to a power of two. This is the provable security property provided by the Congestion-Sensitive BUFL0 algorithm.

This property does not directly imply anonymity. If a particular observation,  $O$ , is only generated by one web page in the world, then an attacker observing  $O$  can conclude with certainty that the victim is visiting that page. To evaluate the security of Congestion-Sensitive BUFL0, we must sample the space of real web sites and confirm that each possible observation can be generated by many different web sites. This is ongoing work.

Finally, note that Congestion-Sensitive BUFL0 does not attempt to hide the fact that the victim is using Congestion-Sensitive BUFL0 and, in the context of censorship circumvention, simply using such a protocol may be sufficient to attract the attention of censors. Note, however, that all traffic analysis defenses must encrypt payload data. Hence, in the current internet where encryption is far from universal, all traffic analysis defenses are easily recognizable, so this problem is not unique to Congestion-Sensitive BUFL0.

## 6. EVALUATION

### 6.1 Web page classifier

Our evaluation examines several factors that may affect the performance of our classifier:

- How do traffic analysis defenses, such as HTTPOS, randomized pipelining, Tor's 512 byte cells, and traffic morphing affect the performance of our classifier?
- How does this compare with other classifiers, such as the Multinomial Naive Bayes classifier of Herrmann, et al. [10] or the SVM classifier of Panchenko, et al. [17]?
- How is performance of our web page classifier affected as the number of web pages goes up?
- How does the size of the training set affect the performance of our web page classifier?
- Does the choice of the web pages in the classification set affect the success rate of our web page classifier?
- Does the state of the browser cache affect the performance of our classifier?

We additionally investigate the overheads of the defense schemes evaluated in this paper.

#### 6.1.1 Experimental Setup

We collected traces using several different computers with slightly different versions of Ubuntu Linux – ranging from 9.10 to 10.10. We used Firefox 3.6.10-3.6.17 and Tor 0.2.1.30, except one computer that used 0.2.2.21-alpha. All Firefox plugins were disabled during data collection. Three of the computers had 2.8GHz Intel Pentium CPUs and 2GB of

<b>DLSVM</b>	Our attack. See Section 3.
<b>Panchenko</b>	Ad hoc SVM classifier of Panchenko, et al. [17], with the libsvm 3.1 implementation from WEKA 3.6.4 and the parameters recommended by Panchenko, et al. ( $c = 2^{17}$ and $\gamma = 2^{-19}$ ).
<b>MNB</b>	The Multinomial Naive Bayes classifier proposed by Herrmann, et al. [10].

**Table 3: The attacks evaluated in our experiments.**

RAM, one computer had a 2GHz AMD Turion Mobile CPU with 2GB of RAM. We scripted Firefox using the Ruby watir-webdriver library and captured packets using tshark, the command-line version of wireshark. For the SSH experiments, we used OpenSSH 5.3p1. Our Tor clients used the default configuration, unless otherwise noted. SSH tunnels passed between two machines on the same local network.

Most of our experiments use data collected from the Alexa Top 1000 web pages. We removed any web pages that failed to load in Firefox (without Tor or any other proxy). If a URL redirected to another location, we replaced it with its redirect target. We then used the top 800 URLs from this cleaned list. We collected traces from each web page in a round-robin fashion. Unless otherwise specified, we cleared the browser cache between each page load. We repeated data collection with four different defense mechanisms, as described below. We collected either 20 or 40 traces from each URL, depending on the defense mechanism in use. We ran most experiments with just the top 100 web pages in our list – we only use full 800 URLs in one experiment to test the scalability of our attack.

This is a “closed-world” evaluation. In such an evaluation, there are only  $k$  web pages in the world. The attacker can collect fingerprints for each page. The victim then chooses one of the pages uniformly at random and loads it in his browser. The attacker observes the victim’s packet trace and attempts to guess which page the victim loaded. Thus, the appropriate metric is the success rate of the attacker, i.e. the percentage of time he guesses correctly. There is no notion of false positive or false negative in this scenario. In contrast, we will evaluate our web site classifier in an open world setting, which does have such considerations.

### 6.1.2 Attacks and Defenses

Table 3 summarizes the attacks evaluated in this paper.

We test each attack against each of the following defenses. For each defense, we also indicate the number of URLs we collected, and the number of visits to each URL. We collected four basic data sets:

**None (SSH) (100x40).** All HTTP traffic is sent through an SSH tunnel.

**SSH + HTTPOS (100x20).** We obtained the prototype implementation that the HTTPOS authors used to evaluate HTTPOS in their paper. Based on some of our early results, they added some additional randomization to their defense. Note that HTTPOS includes both TCP- and HTTP-level defenses. Some web pages caused HTTPOS to crash. We detected crashes and attempted to load the page up to 3 times. If HTTPOS crashed all 3 times, then we added the third, incomplete trace to our data set. Our final data set of

2000 traces contained 33 crash traces, so we do not believe these had a significant effect on our results.

**Tor (800x40).** All HTTP traffic is tunneled through the default Tor configuration. Most experiments only use the top 100 web pages from this dataset.

**Tor + randomized pipelining (100x40).** The Tor project has released a software bundle that includes Tor, the Polipo proxy, and a patched version of Firefox that randomizes the order and pipelining used to load images and other embedded objects in a web page. We use the entire bundle as-is.

We then used these data sets to generate simulations of other defenses, as described below.

**SSH + Sample-based traffic morphing (100x20).** We apply traffic morphing to the traces obtained in the SSH experiment. We morphed all traces to have the same packet size distribution as <http://flickr.com> (selected randomly from our data set). We morphed each direction independently, as described in the traffic morphing paper. To morph a trace, we repeatedly sampled packet sizes from the target distribution and padded (or fragmented) packets in the trace to match the sampled size. Thus our morphed traces have the same packet size distribution as they would under optimal traffic morphing, but the total number of packets transmitted may be higher. The original traffic morphing paper found that optimal traffic morphing and sample-based traffic morphing had equal resilience to attack, so we believe this is a reasonable evaluation of traffic morphing.

**SSH packet count (100x40).** We apply the same transformation to our SSH traces as we did to our Tor traces, as described above.

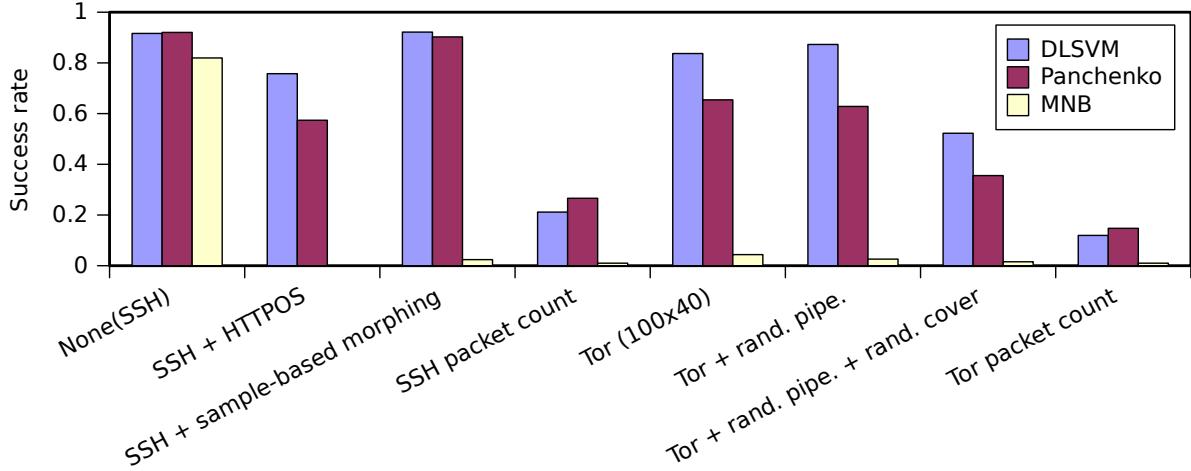
**Tor + randomized pipelining + randomized cover traffic (100x20).** We insert additional cover traffic into the traces collected for the Tor + randomized pipelining experiment. We deleted all packet size information, i.e. traces consisted of only  $\pm 1500$ s. Then, for an input trace of  $l$  packets, we randomly, uniformly, and independently pick  $l$  positions in the trace and insert a 1500 or  $-1500$ , with equal probability, at each position.

**Tor packet count (100x40).** We remove all packet size and direction information from our Tor traces. All that the attacker can observe is the total number of packets transmitted. This experiment explores how much information is revealed by the size of the page being loaded.

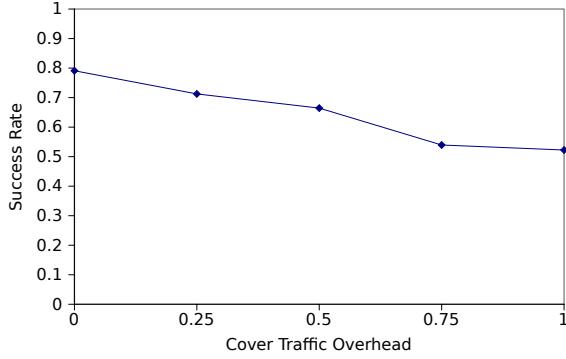
### 6.1.3 Results

We ran each attack against each data set using stratified 10-fold cross validation. Figure 2 shows the results of these experiments. The DLSVM attack generally outperforms the Panchenko and MNB attacks. See Section 7 for discussion.

We performed an experiment to simulate the limits of defenses based on re-ordering, pipelining, padding, and generating extraneous HTTP requests. We added randomized cover traffic and padded all packets to 1500 bytes in the traces in our Tor + randomized pipelining data set, as described above. We varied the cover traffic overhead from 0% to 100%. This experiment is intended to model an idealized version of defenses like randomized pipelining and HTTPOS. Figure 3 shows the influence of adding randomized cover traffic on our attack. With no cover traffic, i.e. with randomized pipelining and packets padded to 1500 bytes, our attack was able to recognize the visited web page almost 80% of the time. If we double the size of the trace by adding ex-



**Figure 2:** Performance of our attack and previously proposed attacks against several proposed defenses.



**Figure 3:** Performance of our attack against Tor with randomized pipelining, all packets padded to 1500 bytes, and varying amounts of cover traffic.

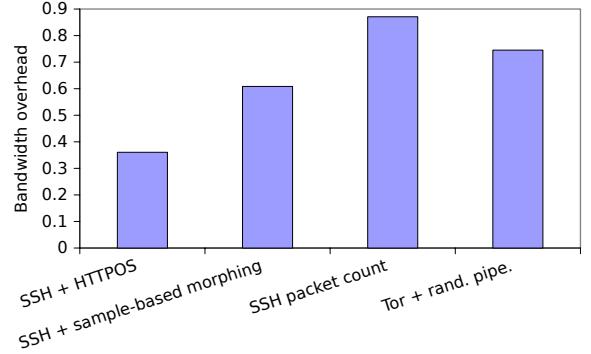
tra cover traffic, our attack can determine the target web page over 50% the time.

Figure 4 shows the bandwidth overheads of the defenses evaluated in this paper. All overheads are normalized to the SSH traces. HTTPS has the lowest overhead, 36%, but is not secure. The other defenses have overhead of over 60% compared to SSH.

Figure 5 shows that the DLSVM, Panchenko, and MNB classifiers work well for both cold cache and warm cache page loads. Although we have not directly evaluated our web page classifier on a mixed cold/warm workload, the web site classifiers evaluated in the next section do use mixed workloads and perform well. Figure 5 also shows that the classifiers perform well on randomly selected web pages loaded through Tor, not just the Alexa top 100 pages.

Figure 6(a) shows how the different attacks perform as the number of web pages they must distinguish increases. Not only does our attack outperform the Panchenko attack when the number of candidate web pages is small, the gap widens as the size of the candidate set increases. For example, our attack can guess which web page, out of 800, that a Tor user is visiting 70% of the time. The Panchenko attack had a success rate of 40% on our set of 800 web pages.

Figure 6(b) shows how additional training data can im-



**Figure 4:** Bandwidth overheads of the defenses evaluated in this paper.

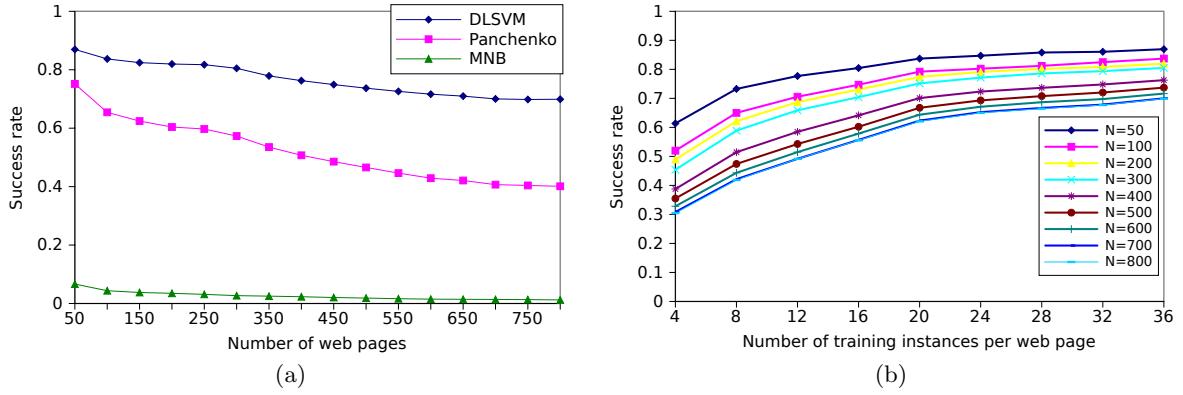
prove the success rate of our attack. Our attack provides satisfactory results, even with a small training set.

## 6.2 Web site classifier

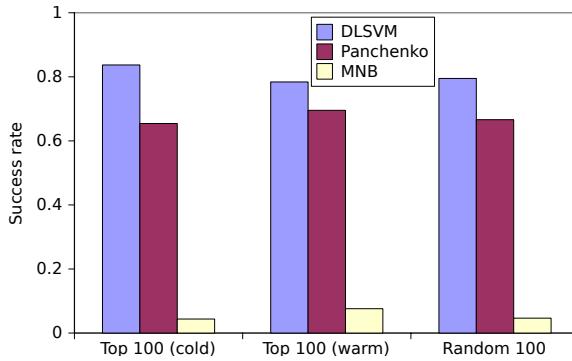
### 6.2.1 Experimental Setup

To evaluate the performance of our web site classifier, we created models for two web sites censored by the Chinese “Great Firewall” – Facebook [7] and IMDB [5] – and constructed page classifiers using the Alexa Top 99 pages, along with the pages in our model for each site. We then collected additional traces for the pages in our models, and ran those traces through the model to compute the probability distribution of classifier outputs for each page in each model, as described in Section 4.

Our Facebook model covers the login page, the user’s home page, and a generic “friend profile page”. It includes warm and cold cache instances of the home and profile pages. Facebook’s home and profile pages use javascript to automatically fetch older items as the user scrolls down the page of past notifications. Our model includes these events. The IMDB model covers the IMDB home page, search results page, movie page, and celebrity page. It includes warm and cold cache states for each page. Transition probabilities between states are artificial for both models – a real



**Figure 6:** (a) Performance of our Tor web page classifiers as a function of the number of possible web pages. (b) Performance of our Tor web page classifier as a function of the training set size.



**Figure 5:** Performance of our web page classifier against Tor under various data collection scenarios.

attacker would derive these from observations of user behavior and would likely have higher accuracy as a result. Initial state probabilities are uniform, since the attacker may begin eavesdropping in the middle of a user’s session. See our technical report for complete specifications of the models[3].

To test our site classifiers, we need traces of the URLs visited by real users. We obtained URL traces for 25 subjects from Eelco Herder. He collected these traces for his empirical study of web user behavior [23]. These traces, from users in Europe, contain numerous visits to IMDB, but no visits to Facebook. Therefore, we have generated artificial traces for Facebook. Our artificial Facebook traces construct visits to Facebook that follow our Facebook model, i.e. we pick a starting Facebook page according to the initial state probabilities of our model, and pick successive pages according to the transition probabilities of our model. We then insert these into real traces so that we create a trace consisting of some Facebook visits and some non-Facebook visits. Since the traces are generated from the same model that the classifier uses, this is obviously an artificial experiment that overestimates the success rate of our attack. However, the IMDB model underestimates the success rate due to the artificial transition probabilities described above, so, together, these two experiments provide rough bounds on the performance of our attack.

We visited the URLs via Tor to generate packet traces that the attacker would observe. Unfortunately, Facebook is

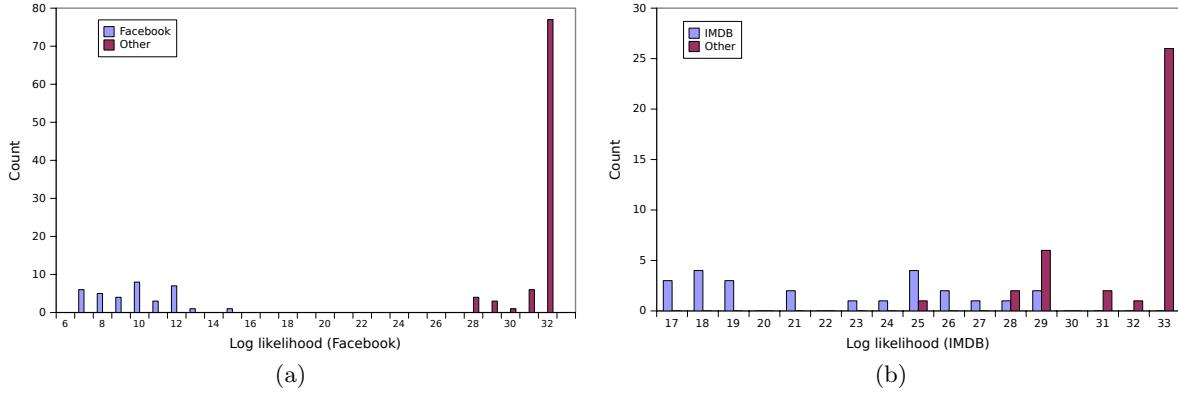
not compatible with Tor’s default configuration. By default, Tor picks a new path every 10 minutes and, to Facebook, the user appears to be coming from the last node in this path. When the path changes, the user appears to have moved from one computer to another – which may be thousands of miles away – in 10 minutes. Facebook detects this and logs the user out. Consequently, Tor users visiting Facebook must alter the Tor configuration to use a fixed path. Thus, we collected all our Facebook data using a fixed Tor path.

### 6.2.2 Results

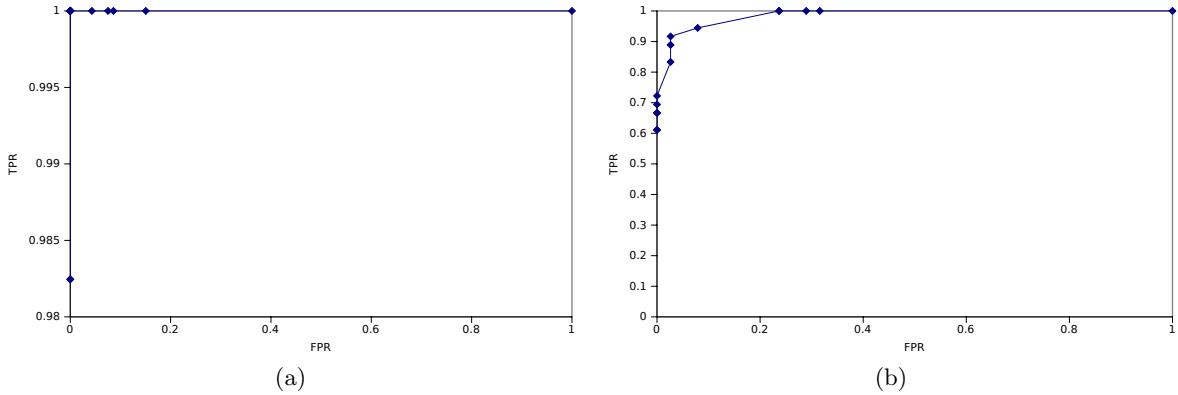
Figures 7(a) and 7(b) show the histogram of log-likelihood scores, under the Facebook and IMDB models, respectively, of 6-page windows of the traces we collected. So, for example, for every window of 6 page loads in the IMDB traces, we ran the packet traces for those 6 page loads through the IMDB model to compute a log-likelihood score. We only considered windows that contained either all IMDB visits or all non-IMDB visits – if a window had, say, 3 IMDB pages and 3 non-IMDB pages, we discarded it from the histogram. As Figure 7(a) shows, the non-Facebook windows are completely separated from the Facebook windows by our model, meaning our classifier works perfectly on this data set. In the IMDB experiment, the non-IMDB windows have, on average, a much higher log-likelihood, indicating that they are not likely to be generated by our IMDB model.

Figure 8 shows the receiver operating curves (ROC) for our Facebook and IMDB classifiers. These curves show the trade-off in False Positive and True Positive rates for varying thresholds of the classifier. As indicated by the histogram in Figure 7(a), the Facebook classifier can achieve 0 false positives and false negatives on our dataset. The IMDB classifier can achieve a 7.9% FP rate and a 5.6% FN rate.

Figure 9 demonstrates how the log-likelihood score correlates with user visits to the target web site over time. Note that these graphs plot traces from multiple browsing sessions – the sessions are separated by gaps in the traces. Only sessions with at least 6 page loads, and at least one page load from the target web site (Facebook or IMDB, respectively), are included in the graphs. The thick, flat, pink line indicates portions of the trace containing page loads from the target web site, page loads from other sites have a thin flat line. The blue lines with markers plot the log-likelihoods of the six-page windows of page loads. As the graphs show, the



**Figure 7:** (a) Distribution of log-likelihood scores (from the Facebook model) for Facebook visits and non-Facebook visits. (b) Distribution of log-likelihood scores (from the IMDB model) for IMDB visits and non-IMDB visits.



**Figure 8:** Receiver operating curves for the (a) Facebook and (b) IMDB web site classifiers.

log-likelihood is below the threshold almost all the time that the user is visiting the target web site, and above the threshold otherwise. An attacker can therefore use our algorithms to pinpoint when a user visits a target web site.

Figure 10 shows anecdotally that our intuition about template matching is correct. We created a set of 99 random web pages and 1 IMDB movie page (Harry Potter). We then ran 100 trials of 4 other IMDB movie pages through the classifier and recorded the pages to which the classifier matched them. The other movie pages matched the Harry Potter movie page 95% of the time, indicating that an attacker can model template pages by using a single instance as a representative of all instantiations of that template.

## 7. DISCUSSION

Our data support several conclusions:

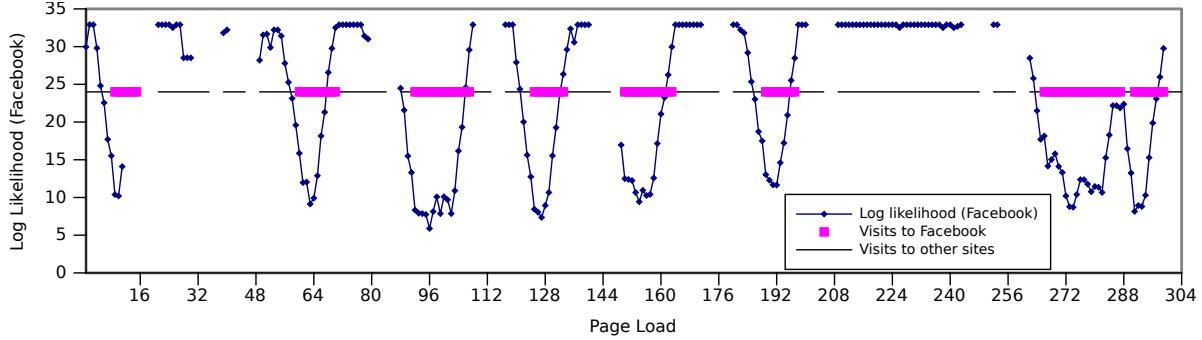
**Existing defenses are inadequate.** Our attack was able to identify the page being loaded over an SSH tunnel with over 90% accuracy. Against Tor, it identified the web page over 80% of the time. The recently proposed randomized pipelining defense did nothing to stop our attack. Our attack is also able to identify web pages loaded over SSH, even if the victim employs traffic morphing or HTTPOS.

**Traffic analysis can infer user actions through several different side channels.** The Panchenko classifier relies primarily on packet sizes and is able to achieve good

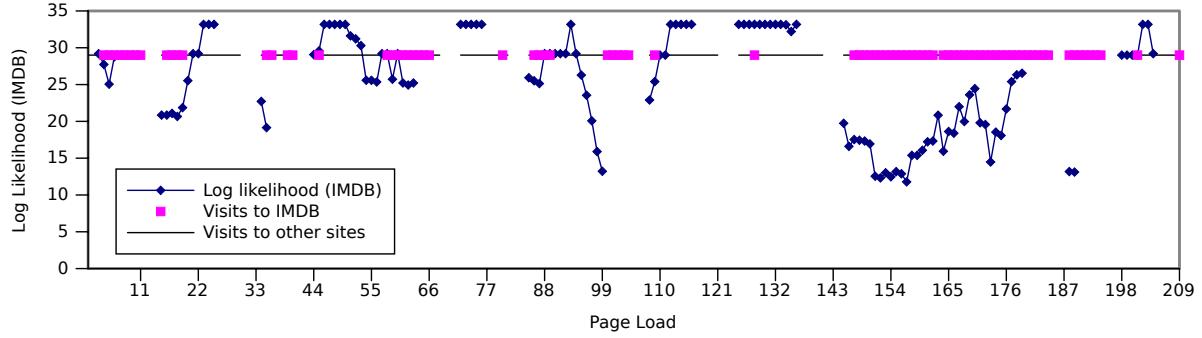
results. On the other hand, our classifier is able to achieve good results even if all packet size information is removed from the trace, as in the randomized cover traffic experiment. Somewhat surprisingly, traffic analysis attacks based solely on the number of packets transmitted (without direction information) can do better than random guessing.

**The DLSVM classifier generally outperforms other classifiers.** It tied or beat the Panchenko classifier in all cases except packet count experiments. Our attack is also much more generic – it does not use ad hoc HTTP-related features. Our page classifier differs from past work primarily in that it does not reduce the packet traces to a fixed-length feature vector. Rather, it passes the trace directly into the classifier. The Damerau-Levenshtein-based classifier is then able to consider multiple aspects of the observation – packet sizes, directions, ordering, etc. – whereas previously-proposed classifiers were only given a finite set of features that had been manually identified by the researchers.

Our experiments suggest that our attack gleans information from several sources, but that the most crucial feature is the pattern of upstream/downstream transmissions. For example, sample-based morphing destroys packet size information, but leaves ordering largely undisturbed. Consequently, our attack works well against morphing. Randomized pipelining destroys some, but not all, ordering information and leaves some packet size information. As a re-



(a)



(b)

**Figure 9:** Log-likelihood scores from the (a) IMDB model and (b) Facebook model for several real traces. Note that the log-likelihood scores are usually below the threshold during visits to the target web site in the trace and above the threshold during visits to other web sites.

sult, our attack is still able to do well. Adding randomized cover traffic and hiding all packet size information obscures the pattern of upstream and downstream transmissions, and hence significantly degrades the performance of our attack. Completely hiding the upstream/downstream information, i.e. reducing the data set to just the number of packets transmitted, almost stops our attack. The Panchenko attack uses packet sizes as its primary feature, but incorporates several ad hoc ordering-based features, so that its performance profile is similar to ours. The MNB classifier has no ordering information, and so its performance drops precipitously when packet size information is obscured.

**Defenses based on randomized requests and cover traffic are not likely to be effective.** In the experiment where we added cover traffic to the Tor + rand. pipe. data, our attack achieved between a 50% and 80% success rate. Furthermore, Figure 3 suggests that additional cover traffic provides diminishing security returns.

**This attack is practical in real settings.** We assume in our evaluation that the victim loads one page at a time and that each page is loaded to completion. This does not always match real user behavior. For example, users may load several pages in different tabs or navigate away from a page before it finishes loading. However, there are two reasons to believe that multiple tabs and similar cover-traffic-based defenses will not protect users. First, our experiments evaluate two different defenses that employ cover traffic. HTTPPOS injects extra HTTP requests into the clients request stream

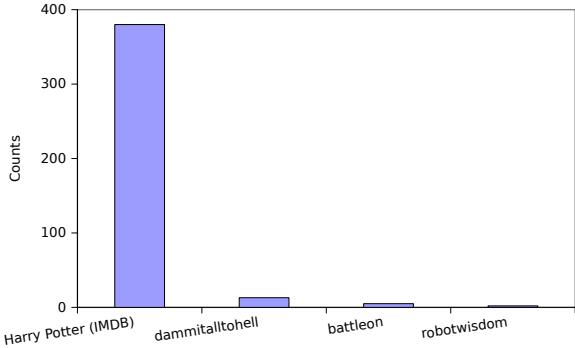
– our attack is still very successful. Similarly, we evaluated Tor with randomized pipelining and with random cover traffic – again, our attack was successful. These two experiments do not evaluate all possible ways of generating cover traffic, but we have yet to find an effective, efficient cover-traffic-based defense. Secondly, a defense scheme should protect users no matter how they surf the web. Even if users do not always load a single page at a time, they do so often enough that it is a valid attack scenario and any defense that fails to protect users in this scenario must be considered broken.

## 8. CONCLUSION

We have demonstrated that Tor is vulnerable to web page and web site fingerprinting attacks. With these attacks, an adversary, such as a local or national government, with the power to monitor a Tor user’s internet connection can infer which web sites the user is visiting. They could use this information to censor the user’s internet connection or to persecute them for visiting banned sites.

Previously proposed defenses, such as traffic morphing, HTTPPOS, and randomized pipelining, impose high costs but do not stop our attack. Consequently, we proposed a new defense with provable security properties, albeit with even higher overhead.

Our attack has several novel features. It is successful even if it ignores packet sizes. Packet sizes have been a crucial feature of almost all prior fingerprinting attacks against Tor and



**Figure 10:** The distribution of matching web pages for various IMDB movie pages. IMDB movie pages almost always match our template sample – the IMDB movie page for Harry Potter. When they didn’t match the Harry Potter page, they always matched one of 3 other web pages out of our 100 distractor pages.

encrypting proxies (e.g. SSH). Although packet size reveals a great deal of information about the data being transferred over a simple encrypting tunnel, Tor conceals this information by padding all data to 512-byte cells. Despite the fact that it ignores packet sizes and uses a simple packet trace comparison method based on the Damerau-Levenshtein distance, its performance on Tor is competitive with a state of the art SVM-based classifier.

We also developed a web site classifier that can use packet traces from a sequence of page loads performed by the victim to infer his online activities. We modeled web sites using HMMs, where each state corresponds to a page or class of pages on the site, and observations are categorized using the classifier developed above.

## Acknowledgments

We thank Daniel Xiapu Luo for providing the HTTPOS source code and invaluable technical support. We thank Eelco Herder for providing us with the URL traces we used to evaluate our web site classifier.

## 9. REFERENCES

- [1] Aurelien Bellet, Amaury Habrard, and Marc Sebban. Good edit similarity learning by loss minimization. *Machine Learning*, 2012.
- [2] George Bissias, Marc Liberatore, David Jensen, and Brian Levine. Privacy vulnerabilities in encrypted http streams. In *Privacy Enhancing Technologies*. 2006.
- [3] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a distance: Website fingerprinting attacks and defenses. Technical Report SPLAT-TR-12-01, Stony Brook University, 2012.
- [4] George Danezis. Traffic analysis of the HTTP protocol over TLS. <http://research.microsoft.com/en-us/um/people/gdane/papers/TLSanon.pdf>.
- [5] The Internet Movie Database. <http://www.imdb.com/>.
- [6] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *Proceedings of the 33rd Annual IEEE Symposium on Security and Privacy*, 2012.
- [7] Facebook. <http://www.facebook.com/>.
- [8] X. Fu, B. Graham, R. Bettati, and W. Zhao. On countermeasures to traffic analysis attacks. In *Information Assurance Workshop*, 2003.
- [9] Xun Gong, Negar Kiyavash, and Nikita Borisov. Fingerprinting websites using remote traffic analysis. In *ACM CCS*, 2010.
- [10] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naive-bayes classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security*.
- [11] Andrew Hintz. Fingerprinting websites using traffic analysis. In *Privacy Enhancing Technologies*. 2003.
- [12] The Internet Society. *Hypertext Transfer Protocol – HTTP/1.1*, 1999.
- [13] Marc Liberatore and Brian Neil Levine. Inferring the source of encrypted http connections. In *ACM CCS*, 2006.
- [14] Liming Lu, Ee-Chien Chang, and Mun Chan. Website fingerprinting and identification using ordered feature sequences. In *ESORICS*. 2010.
- [15] Xiapu Luo, Peng Zhou, Edmond W. W. Chan, Wenke Lee, Rocky K. C. Chang, and Roberto Perdisci. HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows. In *NDSS*, 2011.
- [16] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33:31–88, March 2001.
- [17] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th Workshop on Privacy in the Electronic Society*, 2011.
- [18] Mike Perry. Experimental defense for website traffic fingerprinting. <https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting>, September 2011.
- [19] Yi Shi and Kanta Matsuura. Fingerprinting attack on the Tor anonymity system. In *Information and Communications Security*, volume 5927 of *Lecture Notes in Computer Science*, pages 425–438. Springer Berlin / Heidelberg, 2009.
- [20] Qixiang Sun, Daniel R. Simon, Yi-Min Wang, Wilf Russell, Venkata N. Padmanabhan, and Lili Qiu. Statistical identification of encrypted web browsing traffic. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2002.
- [21] Tor project: Anonymity online. <https://www.torproject.org/>, August 2011.
- [22] Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., 1995.
- [23] Harald Weinreich, Hartmut Obendorf, Eelco Herder, and Matthias Mayer. Not quite the average: An empirical study of web use. *ACM Transactions on the Web*, 1(2):26, 2 2008.
- [24] Andrew M. White, Austin R. Matthews, Kevin Z. Snow, and Fabian Monroe. Phonotactic reconstruction of encrypted VoIP conversations: Hookt on fon-iks. In *Proceedings of the 32nd IEEE Symposium on Security and Privacy*, 2011.
- [25] Charles Wright, Fabian Monroe, and Gerald M. Masson. Hmm profiles for network traffic classification. In *Proceedings of the ACM workshop on Visualization and data mining for computer security*, 2004.
- [26] Charles V. Wright, Scott E. Coull, and Fabian Monroe. Traffic morphing: An efficient defense against statistical traffic analysis. In *NDSS*, 2009.
- [27] Shui Yu, Wanlei Zhou, Weijia Jia, and Jiankun Hu. Attacking anonymous web browsing at local area networks through browsing dynamics. *The Computer Journal*, 2011.
- [28] Fan Zhang, Wenbo He, Xue Liu, and Patrick G. Bridges. Inferring users’ online activities through traffic analysis. In *Proceedings of the Fourth ACM conference on Wireless network security*, 2011.