Stony Brook University

Website Fingerprinting Attacks and Defenses on Anonymity Networks

A Dissertation presented

by

Xiang Cai

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science

Stony Brook University

August 2014

Stony Brook University

The Graduate School

Xiang Cai

We, the dissertation committee for the above candidate for the

Doctor of Philosophy degree, hereby recommend

acceptance of this dissertation

Robert T. Johnson — Dissertation Advisor
Assistant Professor, Department of Computer Science

R. Sekar — Chairperson of Defense
Professor, Department of Computer Science

Phillipa Gill
Assistant Professor, Department of Computer Science

Nikita Borisov
Associate Professor, Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign

This dissertation is accepted by the Graduate School

Charles Taber
Dean of the Graduate School

Abstract of the Dissertation

Website Fingerprinting Attacks and Defenses on Anonymity Networks

by

Xiang Cai

Doctor of Philosophy

in

Computer Science

Stony Brook University

2014

Website fingerprinting attacks [29] enable an adversary to infer which website a victim is visiting, even if the victim uses an encrypting proxy, such as Tor [64]. As a result, researchers have proposed several defenses, most of which focus mainly on hiding packet size information. For example, Tor packs all data into 512-byte cells. Other packet padding schemes include padding to 2^k bytes, or padding all packets to MTU. In 2009, Wright, et al., proposed traffic morphing, which alters the size of the packets transmitted so that the packet size distribution appears to be from a different web page [73]. Recently researchers proposed several application-level defenses against traffic analysis attacks, including HTTPOS [43] and randomized pipelining over Tor [53].

We present a novel web page fingerprinting attack DLSVM, that is able to defeat these defenses. 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 have shown that all these defenses are ineffective, and strongly suggest that ad hoc defenses against traffic analysis are not likely to succeed. Therefore, we develop a theoretical model of website fingerprinting attacks and defenses and use it to prove several

results. First, we develop bounds on the trade-off between overhead and security that any fingerprinting defense can achieve. This enables us to compare schemes with different overhead/security trade-offs by comparing how close they are to optimal. We then propose, implement, and evaluate a new defense scheme, which we call Congestion-Sensitive BuFLO, based on the BuFLO defense proposed by Dyer, et al. [20]. Our experiments find that Congestion-Sensitive BuFLO has high overhead (around 2.3-2.8x) but can get 6× closer to the overhead/security trade-off lower bound than Tor or plain SSH.

Lastly, our theoretical analysis suggests that the reason website fingerprinting defenses are expensive is not because websites are so different; it is because defenses lack the knowledge of where to put cover traffic, so they have to put it everywhere. We propose a provably secure defense Glove, and demonstrate that this defense can defeat an ideal attacker while providing better overhead/security trade-off than previously proposed defenses.

Dedicated to my family.

Table of Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to express my heartfelt gratitude to my advisor, Rob Johnson, for his guidance and support during my Ph.D. studies at Stony Brook University. He has always been an innovative researcher, a patient mentor, and a great friend. As a teacher, he taught me how to be a good researcher; as a friend, he showed me how to be a better person. I am lucky enough to have Rob as my advisor during this long but wonderful Ph.D. journey.

Next, I would like to thank my dissertation committee members, Phillipa Gill, R. Sekar and Nikita Borisov, for their valuable suggestions and comments to my thesis and research.

I have also learned a lot during my Internship at HP Labs. I would like to thank my mentor Pratyusa Manadhata and my manager Bill Horne, for the great internship experience they provided.

I have had the honor to work with many talented colleagues in SPLAT lab: Michael Hart, Yuwei Ethan Gui, Jianing Sandra Guo, Xincheng Zhang, Jun Yuan and Rishab Nithyanand. I am also fortunate to meet so many friends here at Stony Brook. To name just a few of them: Xiaomeng Ban, Congcong Che, Chen Pan, Tianyuan Wu, Yang Liu, Ping Cao, Lin Chen, Xiaoyang Gong, Yi Shang, Yifan Peng, Ziyi Zheng, Tingbo Hou and Min Lu.

Last but not least, I would like to thank my family, especially my parents and my lovely wife, for the love and faith they gave me. I never could have made it this far without their support and encouragement. Thank you and I love you!

Chapter 1

Introduction

Anonymous communication systems hide the identities of the parties involved in the communication by providing *unlinkability* between messages and their senders and receivers, thus giving *anonymity* to end users. A set of definitions for *unlinkablity* and *anonymity* were proposed by Pfitzmann, et al. in 2008 [55]. Chaum et al. introduced the Mix networks in 1981 [13], which use a set of trusted servers to relay messages from senders to receivers while providing perfect anonymity as long as at least one of the servers is trustworthy. Based on Mix networks, Goldschlag, et al. proposed the idea of onion routing [26], which is employed by Tor [64], the most popular anonymity network on the Internet today. Other anonymity networks include JAP [34], PipeNet [14] and Freedom [12]. Anonymous web browsing systems, such as Tor, hide users' browsing activities from outside observers. Most anonymity networks achieve anonymity by directing a user's web traffic through a proxy or chain of proxies before it reaches the web server. The traffic between the user and the proxies is encrypted, so that an outside observer cannot infer users' activities from packet contents.

People use anonymity networks for many reasons. For example, activists and reporters use anonymity networks to report from danger zones; researchers can use anonymity networks to anonymously research sensitive topics that may be banned in certain countries. More importantly, users can use anonymity networks to visit websites banned by Internet censors [2]. In this case, anonymity networks are used to evade the censor's automated blocking, and to hide the user's activities that might have legal or political repercussions.

Although many web browsing privacy mechanisms, such as SSL, Tor, and encrypting tunnels, encrypt the data transferred between the user and the proxies, they do not effectively hide other information, such as packet sizes, timing, and directions of packets. These information allows an adversary to mount website fingerprinting attacks by performing traffic analysis on the observed traffic. In a website fingerprinting attack, an adversary analyzes these features and attempts to infer the web page being visited by a client. Website finger-

printing attackers are passive attackers, so they do not alternate the traffic and hence are difficult to detect.

Researchers have proposed various website fingerprinting attacks and defenses, and they have shown that web page fingerprinting attacks are possible against many privacy services, including IPSec tunnels, SSH tunnels, and Tor [64, 28, 50, 20, 40].

As a result, researchers have proposed several defenses, primarily aimed at hiding packet size information. For example, 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 [73]. At the 2012 Oakland conference, Dyer, et al. [20] showed that an attacker could infer, with a success rate over 80%, which of 128 pages a victim was visiting, even if the victim used network-level countermeasures. They also performed a simulation-based evaluation of a hypothetical defense, which they call BuFLO, and found that it required over 400% bandwidth overhead in order to get the success rate of the best attack down to 5%, which is still well-above the ideal 0.7% success rate from random guessing.

Researchers have recently proposed defenses based on manipulating the sequence and structure of the HTTP requests generated by the browser. HTTPOS, published at NDSS 2011, manipulates TCP MSS and window size parameters to obscure packet sizes, but also includes several HTTP-specific mechanisms [43]. For example, HTTPOS 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 HTTPOS, 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 [53].

In this dissertation, we show that:

- All existing defenses are either not effective or not efficient.
- There is likely substantial room for improvement.
- Defenses can substantially increase their efficiency if they have information about the websites users are likely to visit.

We propose an attack called DLSVM, which is very effective against a wide variety of defenses, including HTTPOS, randomized pipelining, and several other defenses. Table 1.1

Defense	Rate
None (SSH tunnel)	91.6%
SSH + HTTPOS	75.7%
SSH + Sample-based morphing	92.1%
Tor	83.7%
Tor + randomized pipelining	87.3%
Tor + random pipelining + random traffic	52.2%

Table 1.1: Success rate of our web page fingerprinting attack against each defense evaluated in Chapter 3.3. The success rate is the probability that the attack was able to correctly guess which of 100 web pages the victim was visiting.

summarizes the results of our attack on each 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 HTTPOS or sample-based traffic morphing.

We also evaluated our attack against a simulated Tor implementation that uses randomized pipelining, pads all packets to 1500 bytes, and adds 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.

Attack	Defense	Database Size	Success Rate
DLSVM attack	Tor	100	**83.7%**
Ad hoc SVM [50]	Tor	100	65.4%
Cosine Similarity [61]	Tor	20	50%
Multinomial Naive-Bayes [28]	Tor	100	4.4%
DLSVM attack	Tor + random pipelining	100	**87.3%**
Ad hoc SVM [50]	Tor + random pipelining	100	62.8%
DLSVM attack	None (SSH)	100	91.6%
Ad hoc SVM [50]	None (SSH)	100	**92.0%**
Multinomial Naive-Bayes [28]	None (SSH)	100	81.9%

Table 1.2: Success rates of DLSVM attack compared to relevant previous attacks. The results for the cosine similarity classifier are taken from Shi, et al [61]. All other results are computed using our implementations on our data sets.

Ours is the first demonstration that application-level defenses, such as HTTPOS and randomized pipelining, are not effective. 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 1.2. In 2009, Herrmann, et al., proposed a fingerprinting attack based on a Multinomial Naive-

Bayes classifier [28], which, in our experiments is able to identify the 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 [61], 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 achieves only a 65% success rate on our data set [50]. 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. [20].

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 [68] 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 show that existing network and application-level defenses also fail to provide sufficient protection. As a result, it is not currently known whether there exists any efficient and secure defense against website fingerprinting attacks. We investigate this issue further, and propose Congestion-Sensitive BuFLO, which extends Dyer's BuFLO scheme to include congestion sensitivity and rate adaptation. CS-BuFLO is TCP-friendly and adapts its transmission rate dynamically, which makes it very practical to deploy. This also poses a challenge: adapting too quickly to the website's transmission rate can reveal information about which website the victim is visiting. CS-BuFLO balances these performance and security constraints by limiting the rate and granularity of adaptation. We also introduce

Defense	n	Method	Source	Panchenko	VNG++	DLSVM	BW Ratio	Latency Ratio
CS-BuFLO (CTSP)	200	Empirical	Chapter 5	18.0	13.0	20.6	2.796	3.271
CS-BuFLO (CPSP)	200	Empirical	Chapter 5	24.2	16.5	34.3	2.289	2.708
CS-BuFLO (CTSP)	120	Empirical	Chapter 5	23.4	20.9	28.9	2.799	3.444
CS-BuFLO (CPSP)	120	Empirical	Chapter 5	30.6	22.5	40.5	2.300	2.733
BuFLO $(0, 40, 1000)$ [a]	128	Simulation	[20]	27.3	22.0	N/A	1.935	N/A
BuFLO $(0, 40, 1500)$	128	Simulation	[20]	23.3	18.3	N/A	2.200	N/A
BuFLO $(0, 20, 1000)$	128	Simulation	[20]	20.9	15.6	N/A	2.405	N/A
BuFLO $(0, 20, 1500)$	128	Simulation	[20]	24.1	18.4	N/A	3.013	N/A
BuFLO $(10^5, 40, 1000)$	128	Simulation	[20]	14.1	12.5	N/A	2.292	N/A
BuFLO $(10^5, 40, 1500)$	128	Simulation	[20]	9.4	8.2	N/A	2.975	N/A
BuFLO $(10^5, 20, 1000)$	128	Simulation	[20]	7.3	5.9	N/A	4.645	N/A
BuFLO $(10^5, 20, 1500)$	128	Simulation	[20]	5.1	4.1	N/A	5.188	N/A
HTTPOS	100	Empirical	Chapter 3	57.4	N/A	75.8	1.361	N/A
Tor+random pipelining	100	Empirical	Chapter 3	62.8	N/A	87.3	1.745	N/A
Tor	100	Empirical	Chapter 3	65.4	N/A	83.7	N/A	N/A
Tor	120	Empirical	Chapter 3	56.3	36.8	77.4	1.247	4.583 [b]
Tor	200	Empirical	Chapter 3	50.1	31.8	75.1	1.244	4.919
Tor	775	Empirical	[50]	54.6	N/A	N/A	N/A	N/A
Tor	800	Empirical	Chapter 3	40.1	N/A	50.6	N/A	N/A
SSH	120	Empirical	Chapter 3	86.5	75.0	80.7	1.128	1
SSH	200	Empirical	Chapter 3	84.4	72.9	79.4	1.111	1

[a] The triplet (x, y, z) represents the three parameters used by BuFLO, where $\tau = x, \rho = y, d = z$.

[b] Note that the high latency of TOR is largely due to its onion routing protocols — a cost that other defenses do not incur.

Table 1.3: Main evaluation results for CS-BuFLO, and comparison to results on other schemes reported by other researchers.

a stream padding scheme that exploits interaction between the client and server to provide better security at lower bandwidth than simpler stream padding schemes that operate independently in each direction.

We present a complete specification of CS-BuFLO, and have implemented CS-BuFLO in a custom version of OpenSSH. Our implementation also includes a Firefox browser plugin that informs the SSH client when the browser has finished loading a web page. The CS-BuFLO implementation uses this information to reduce the amount of padding performed after the page load has completed.

We evaluate CS-BuFLO, and compare it to Tor on the Alexa top 200 websites in a closed-world setting. The Alexa top 200 websites represent approximately 91% of page loads on the internet [3], so these results reflect the security users will obtain when using these schemes in the real world. Furthermore, prior work on website fingerprinting attacks has found that an attacker's success rate only decreases as the number of websites increases, so our results

give high-confidence upper bounds on the success rate these attacks may achieve in larger settings.

In our experiments, CS-BuFLO uses 2.8 times as much bandwidth as SSH (i.e. no defense) and the best known attack had only a 20% success rate at inferring which of 200 websites a victim was visiting. This is a substantial improvement over previously proposed schemes — the same attack had a success rate over 75% against Tor and SSH under the same conditions.

Table 1.3 compares our results with results reported by other researchers. These comparisons must be done carefully, since the experiments used different numbers of websites and methodologies. Nonetheless, the following conclusions are clear from the data:

- CS-BuFLO hides more information than Tor, SSH, HTTPOS, and Tor with randomized pipelining, albeit with higher cost. For example, the DLSVM attack has a lower success rate against CS-BuFLO in a closed-world experiment with 100 websites than it has against Tor with 800 websites.

- Overall, CS-BuFLO achieves approximately the same overhead/security trade-off in our empirical analysis as BuFLO achieved in Dyer's simulated evaluation. For example, CS-BuFLO in CTSP mode had a bandwidth ratio of 2.8 and Panchenko's attack had a success rate of 23.4% on 120 websites. BuFLO with $\tau = 0$, $\rho = 40$, and $d = 1500$ had almost identical security, but a bandwidth ratio of 2.2. Although CS-BuFLO optimizes many aspects of the BuFLO protocol, an empirical evaluation presents issues that do not arise in a simulation, such as dropped packets, retransmissions, and application-level timing dependencies.

These results show that CS-BuFLO achieves better overhead/security trade-off than existing real defense systems, making it a promising technology to be deployed in the real world.

In addition to the empirical work on CS-BuFLO, we provide an analytical study of the problem of defending against website fingerprinting attacks. We show that constructing an optimally-efficient defense scheme for a given set of websites is an NP-hard problem. We then develop lower bounds on the best possible trade-off between overhead and security that any website fingerprinting defense can achieve. Specifically, given a set of websites and a desired security level, we can compute a lower bound on the bandwidth overhead that any defense scheme with that security level can incur on those websites. This enables us to compare defenses that offer different overhead/security trade-offs by comparing how close they are to the lower bound.

Our study also shows how to compute the open-world performance of an attack based on its performance in a closed-world experiment. Thus, researchers can evaluate attacks

and defenses in the simpler closed-world model and, using our method, compute open-world performance results. We also use the closed-world/open-world connection to investigate the danger that fingerprinting attacks pose in the real world. We find that, without any defense whatsoever, fingerprinting attacks can pose a significant threat to visitors to popular web pages. For example, an ideal attacker against defenseless victims could recognize visits to the 100 most popular websites with a false discovery rate of less than 50%.

Our theoretical analysis suggests that a small amount of well-placed cover traffic can make many websites look similar. This conclusion is based on the assumption that a defender knows the sizes of protected websites in advance, so she can pad each website to the same size. Although in reality, hiding only the sizes of websites is not enough to defeat a powerful attacker, the idea of learning information from protected websites in advance is a valuable insight. Based on this idea, we propose a provably secure defense scheme, which we call Glove. To protect n websites, Glove collects traces of popular websites and computes a transcript, z, of packet sizes and timings that it replays whenever a user loads one of the websites. Since the same observation z is always presented, Glove bounds the success rate of any attacker to $\frac{1}{n}$.

In summary, our work makes the following contributions:

- We show that recently proposed application-level defenses, such as HTTPOS 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 and give a complete specification of the CS-BuFLO protocol, describing optimizations to make the protocol congestion sensitive, rate adaptive, and efficient at hiding macroscopic website features, such as total size and the size of the last object.
- We implement a prototype of the defense in SSH, which also includes a Firefox plugin to notify the proxy when the browser finishes loading a web page.
- We present empirical evaluation results for CS-BuFLO, Tor, and SSH, and shows that CS-BuFLO provides better security, albeit at higher bandwidth costs. We also show that CS-BuFLO is closer to the lower bound on the overhead/security trade-off than Tor and SSH.

- We provide the first analytical results on the website fingerprinting defense problem, showing that constructing an optimal defense is NP-hard and developing lower bounds on the best possible trade-off between overhead and security.

- We show the connection between the two most widely used evaluation models: open-world and closed-world.

- We propose and evaluate a provably secure defense, Glove, which achieves a better overhead/security trade-off by using information collected offline about the websites users are likely to visit.

The remainder of the dissertation is organized as follows. Chapter 2 describes the background of website fingerprinting attacks and defenses. In Chapter 3 we describe and evaluate our website fingerprinting attack, DLSVM. Chapter 4 presents our analytical results. Chapter 5 and Chapter 6 describe and evaluate CS-BuFLO and Glove, respectively. We then discuss related issues and conclude the dissertation in Chapter 7.

Chapter 2

Background

2.1 Problem Description

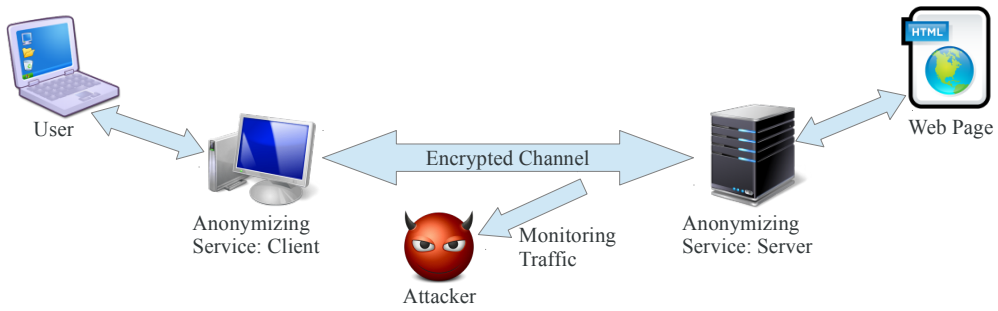


Figure 2.1: Website fingerprinting attack threat model.

In a website fingerprinting attack, an adversary is able to monitor the communications between a victim's computer and an anonymizing service server, as shown in Figure 2.1. The service server may be an SSH proxy, VPN server, Tor, or other privacy service. The traffic between the user and server is encrypted, so the attacker can only see the timing, direction, and size of packets exchanged between the user and the server. Based on this information, the attacker attempts to infer the website(s) that the user is visiting via the server. The attacker can prepare for the attack by collecting information about websites in advance. For example, he can visit websites using the same privacy service as the victim, collecting a set of website "fingerprints", which he later uses to recognize the victim's site.

Website fingerprinting attacks are an important class of attacks on private browsing systems. For example, Tor states that it "prevents anyone from learning your location or browsing habits."[64] Successful fingerprinting attacks undermine this security goal. Fingerprinting attacks are also a natural fit for governments that monitor their citizens' web

browsing habits. The government may choose not to (or be unable to) block the privacy service, but nonetheless wish to infer citizens' activities when using the service. Since it can monitor international network connections, the government is in a good position to mount website fingerprinting attacks.

Website fingerprinting defenses try to erase the "fingerprints" by reshaping websites' page load traffic in various ways. Some defenses try to generate different traffic for a website on each page load [53, 43], and others try to make the page load traffic similar for a group of websites [73, 20].

2.2 Evaluation Models

Researchers have proposed two models for evaluating website fingerprinting attacks and defenses: the closed-world model and the open-world model. A closed-world model consists of a finite number, n, of web pages. Typical values of n used in past work range from 100 to 775 [20, 50]. The attacker can collect traces and train his attack on the websites in the world. The victim then selects one website uniformly at random, loads it using some defense mechanism, such as Tor or SSH, and the attacker attempts to guess which website the victim loaded. The key performance metric is the attacker's average success rate.

In an open-world model, there is a population of victims, each of which may visit any website in the real world, and may select the website using a probability distribution of their choice. The attacker does not know any individual victim's distribution over websites, but has aggregate statistics about website popularity. The attacker's goal is to infer which of the victims are visiting a particular "website of interest", i.e. an illegal or censored site. In this case, the primary evaluation criteria are false positives and false negatives.

The relationship between these two models are discussed in Chapter 4.2.

2.3 Related Work

2.3.1 Attacks

Researchers have studied attacks on anonymity systems from a variety of angles [33, 58]. Some attacks are designed to exploit vulnerabilities of a specific anonymity system. For example, Zhang exploits a vulnerability in Tor authentication protocol during concurrent runs [77]; McLachlan, et al., reveals the IP addresses of Tor bridges due to several bridge related architectural vulnerabilities in Tor [45]. These attacks are not generic and can be addressed by fixing implementation bugs. Some attack model involves a global passive

adversary, who is capable of observing large portion or all of the network traffic [21, 49]. However, such an attacker rarely exists in reality, so this attack model is not considered by some anonymity networks [18].

Some attacks focus on discovering the identity of the anonymous network user, others focus on discovering the servers they interact with. Some active attacks require subverting nodes in the anonymity network and injecting traffic into the network [7, 19, 39, 56, 46, 51, 22, 11]. An Attacker can increase the probability that their malicious nodes are chosen during the circuit set up phase, and enabling them to uncover the victim's route through the anonymizing network [7, 51]. Injecting traffic allows an attacker to mount a denial of service attack, and the attack forces the messages in the network to be retransmitted, thus giving more opportunities to an attacker [11]. An attacker can also subvert web servers visited by anonymous users [66].

Traffic and time analysis is a useful technique for network attackers. It deduces information from network traffic by analyzing communication patterns. It is widely used in the research field of network security [57, 5, 52, 70, 78, 36, 35]. There are also lots of such attacks related to anonymity networks [6, 62, 65, 59, 44, 47, 16, 4, 31, 41, 74].

We focus on one particular type of traffic analysis attacks, web page fingerprinting attacks, which assume a passive local eavesdropper and focus on de-anonymizing the identities of the web pages a user is visiting. 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 evade 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.

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 [10, 15, 27, 28, 29, 40, 42, 63, 75, 76, 20]. 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 [20]. They did not evaluate application-level defenses, such as HTTPOS or randomized pipelining.

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 [28]. 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 [61]. 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 used by Herrmann, et al., [50, 28]. We re-implemented their attack and obtained a 65.4% success rate on our data set of 100 web pages.

The unpublished work of Danezis [15] 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 [42], but their fingerprints depend heavily on packet size information, which is not available when attacking Tor users. Yu, et al. [75] 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.

2.3.2 Defenses

Network-level website fingerprinting defenses pad packets, split packets into multiple packets, or insert dummy packets. Dyer, et al., list numerous approaches to padding individual packets, including pad-to-MTU, pad-to-power-of-two, random padding, etc.[20]. They showed that none of the padding schemes was effective against the attacks they evaluated. Wright, et al., proposed traffic morphing, in which packets are padded and/or fragmented so that they conform to a specified target distribution[73]. Dyer, et al., defeated this defense, as well[20]. Lu, et al., extended traffic morphing to operate on n-grams of packet sizes, i.e. their scheme pads and fragments packets so that n-grams of packet sizes match a target distribution[42]. Dyer, et al. also proposed BuFLO, which pads or fragments all packets to a fixed size, sends packets at fixed intervals, injecting dummy packets when necessary, and always transmits for at least a fixed amount of time[20]. They found that they could reduce their best attack's success rate to 5% (when guessing from 128 websites), at a cost of 400% bandwidth overhead. Fu, et al., found in early work that changes in CPU load can cause slight variations in the time between packets in schemes that attempt to send packets at fixed intervals, and recommended randomized inter-packet intervals instead[24].

Application-level defenses alter the sequence of HTTP requests and responses to further obfuscate the user's activity. At NDSS 2011, Luo, et al., described HTTPOS, a collection of HTTP- and TCP-level tricks for fooling traffic analysis attacks previously described in the literature [43]. 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 HTTPOS.

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 [53]. 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.

2.3.3 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 [72], 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 [71].

Chapter 3

DLSVM Attack

In this chapter, we present a powerful website fingerprinting attack, DLSVM, which defeats HTTPOS, randomized pipelining, and several other defenses. Developing an attack helps us to understand the vulnerabilities of existing defenses, and gives us valuable insights towards building better defenses.

3.1 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 [32]. 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 ℓ packets as a vector $t = (d_1, \ldots, d_\ell)$, where $d_i = \pm s_i$, where s_i is the size of the ith packet and the sign indicates the direction of the packet. Our attack compares traces t and t' using the Damerau-Levenshtein edit distance [48], 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. [8].

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 Chapter 3.3, we deleted all packet size information, i.e. traces were reduced to sequences of $\pm$1s.

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 [67] using a kernel based on edit distance:

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

The γ 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 ± 1s 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 $t^* = \text{argmin}_{t'} L(t, t')$ 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 ℓ^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.

3.2 Recognizing Web Sites

As the evaluation results in Chapter 3.3 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 [68] 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 ℓ additional traces and estimates $\Pr[o|s]$ as the fraction of the ℓ 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 Chapter 3.3 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 $s_0, \ldots, s_r$.

State s_0 corresponds to the initial page load, and states $s_1, \ldots, s_r$ 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 $s_0, \ldots, s_r$, and transitions from the s_is 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.

3.3 DLSVM Evaluation

3.3.1 Web Page Classifier

Our evaluation examines several factors that may affect the performance of DLSVM 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. [28] or the SVM classifier of Panchenko, et al. [50]?
- 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.

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 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.

DLSVM	Our attack. See Chapter 3.1.
Panchenko	Ad hoc SVM classifier of Panchenko, et al. [50], with the libsvm 3.1 implementation from WEKA 3.6.4 and the parameters recommended by Panchenko, et al. ($c = 2^{17}$ and $\gamma = 2^{-19}$).
MNB	The Multinomial Naive Bayes classifier proposed by Herrmann, et al. [28].

Table 3.1: The attacks evaluated in our experiments.

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.

Table 3.1 summarizes the attacks evaluated here.

Attacks and Defenses. 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 remove all packet size and direction information from our SSH 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.

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 ± 1500s. 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 apply the same transformation to our Tor traces as we did to our SSH traces, as described above.

Results. We ran each attack against each data set using stratified 10-fold cross validation. Figure 3.1 shows the results of these experiments. The DLSVM attack generally outperforms the Panchenko and MNB attacks. See Chapter 7 for more discussions.

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 3.2 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.

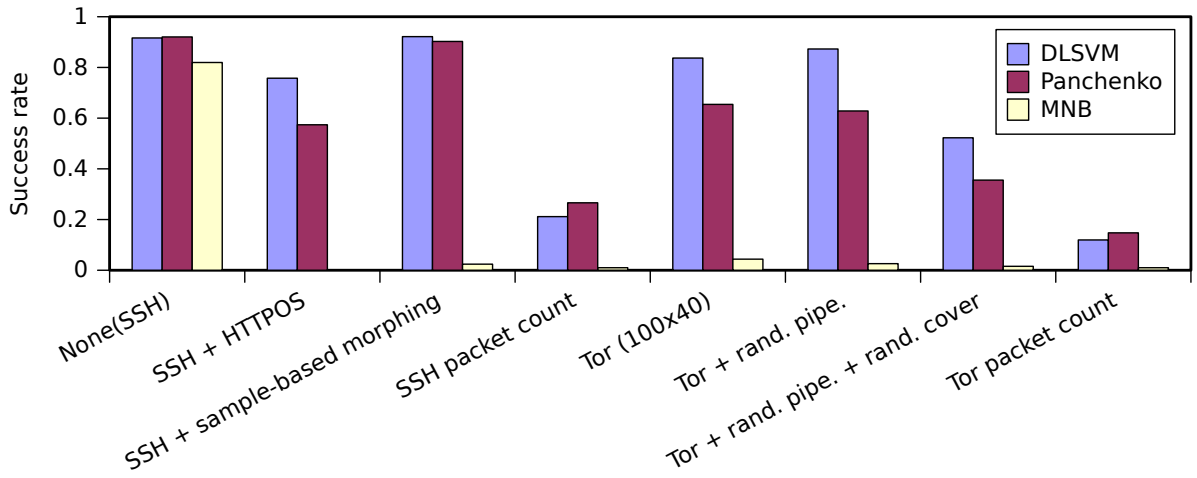


Figure 3.1: Performance of DLSVM and previously proposed attacks against several proposed defenses.

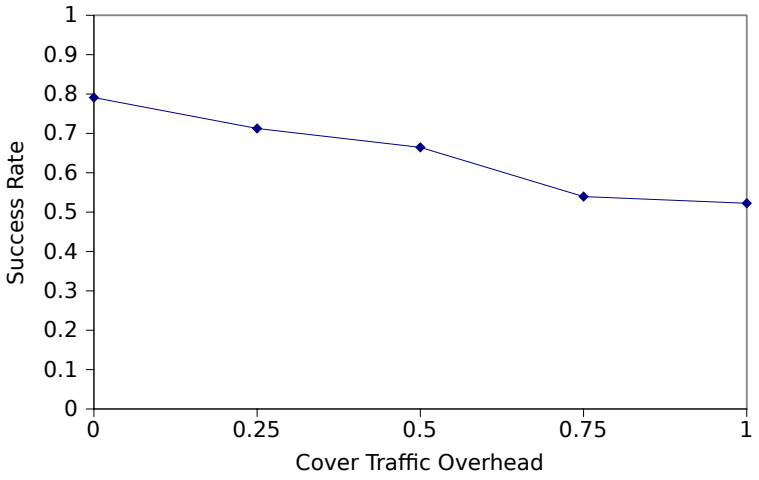


Figure 3.2: Performance of DLSVM against Tor with randomized pipelining, all packets padded to 1500 bytes, and varying amounts of cover traffic.

If we double the size of the trace by adding extra cover traffic, our attack can determine the target web page over 50% the time.

Figure 3.3 shows the bandwidth overheads of the defenses evaluated in this section. All overheads are normalized to the SSH traces. HTTPOS has the lowest overhead, 36%, but is not secure. The other defenses have overhead of over 60% compared to SSH.

Figure 3.4 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 3.4 also shows that the classifiers perform well on randomly selected web pages loaded through Tor, not just the Alexa top

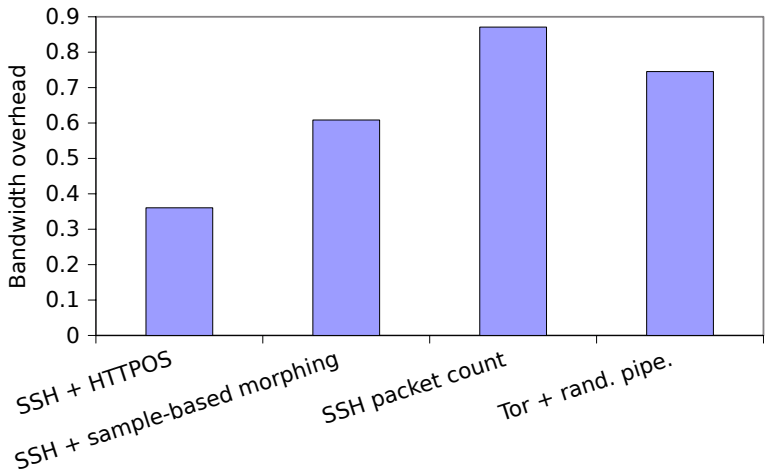


Figure 3.3: Bandwidth overheads of the defenses evaluated in this section.

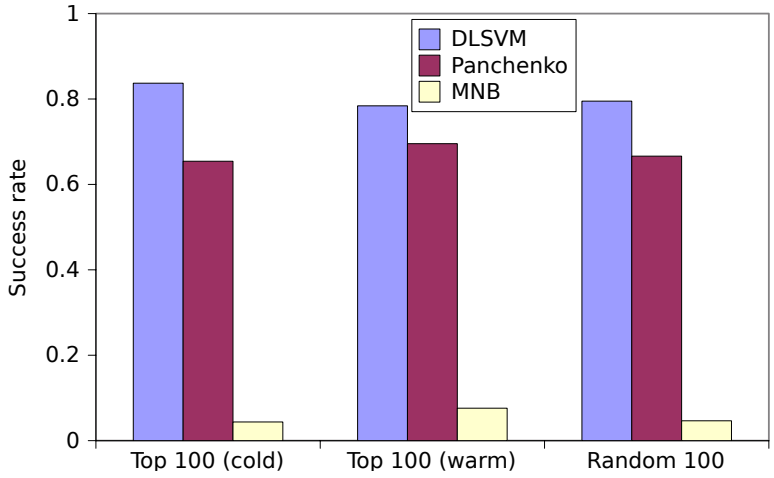
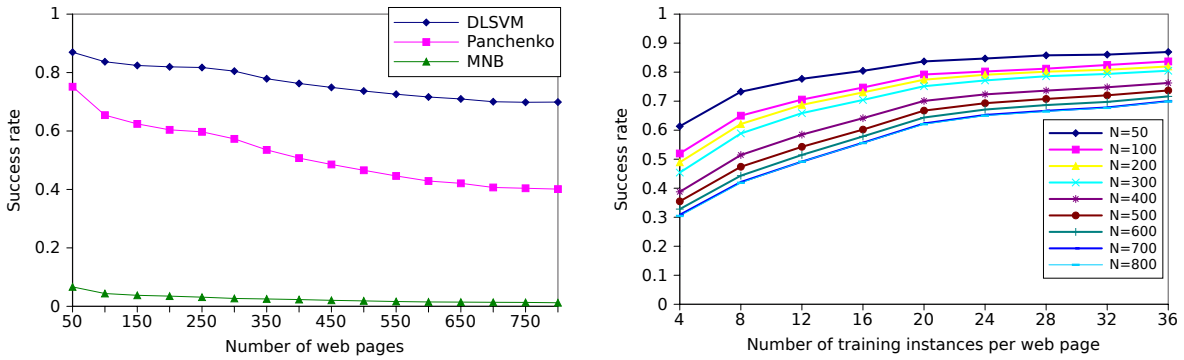


Figure 3.4: Performance of DLSVM against Tor under various data collection scenarios.

100 pages.

Figure 3.5(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 3.5(b) shows how additional training data can improve the success rate of our attack. Our attack provides satisfactory results, even with a small training set.



(a) (b)

Figure 3.5: (a) Performance of DLSVM against Tor 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.

3.3.2 Web Site Classifier

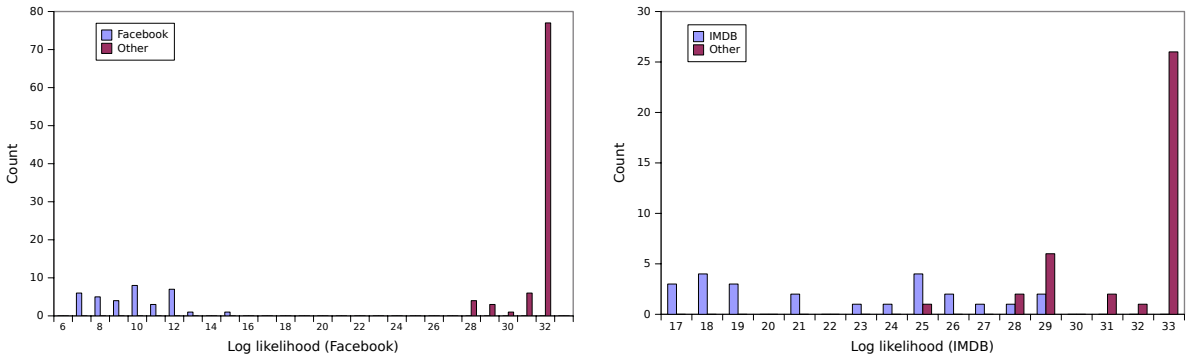
Experimental Setup. To evaluate the performance of our web site classifier, we created models for two web sites censored by the Chinese "Great Firewall" – Facebook [23] and IMDB [17] – and constructed page classifiers using the Alexa Top 99 pages, along with the pages in our model for each site. We then 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 Chapter 3.2.

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 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 Appendix A. HMMs for Facebook and IMDB for complete specifications of the models.

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 [69]. 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.



(a) (b)

Figure 3.6: (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.

Results. Figures 3.6(a) and 3.6(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 3.6(a) shows, the non-Facebook windows are completely separated from the Facebook windows by our model, meaning our classifier works perfectly on this

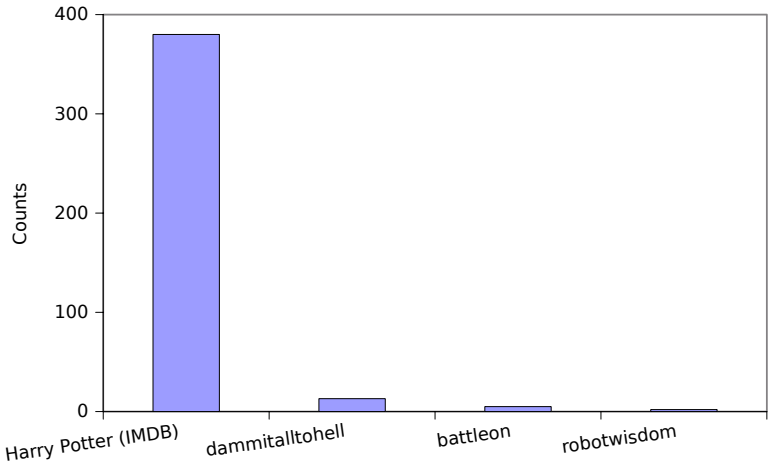
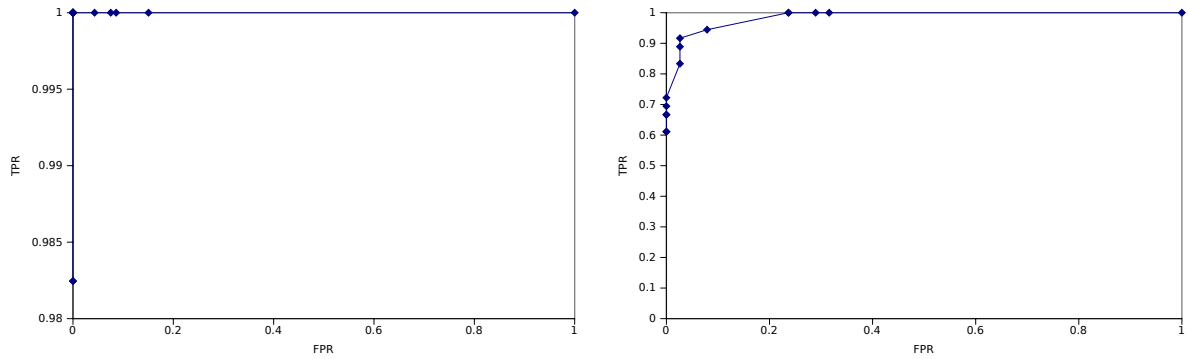


Figure 3.7: The distribution of matching web pages for various IMDB movie pages. IMDB movie pages almost always match our template sample – the IMDB movie page for Harry Potter. When they didn't match the Harry Potter page, they always matched one of 3 other web pages out of our 100 distractor pages.

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.

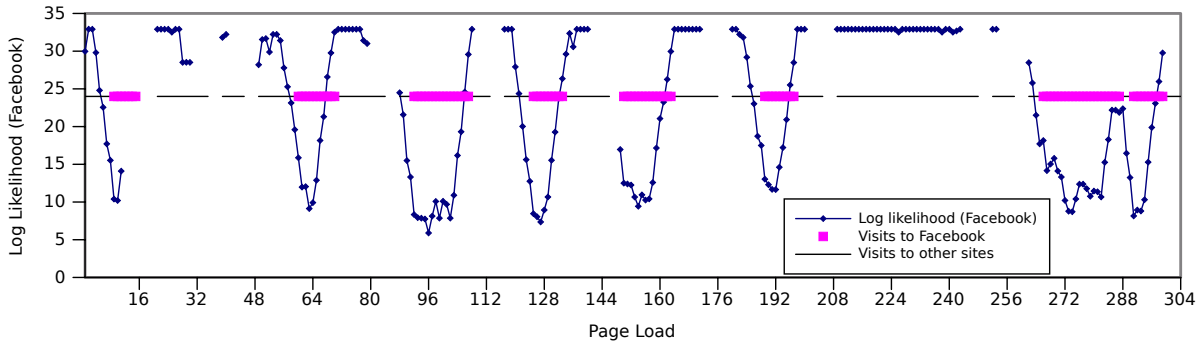


(a) (b)

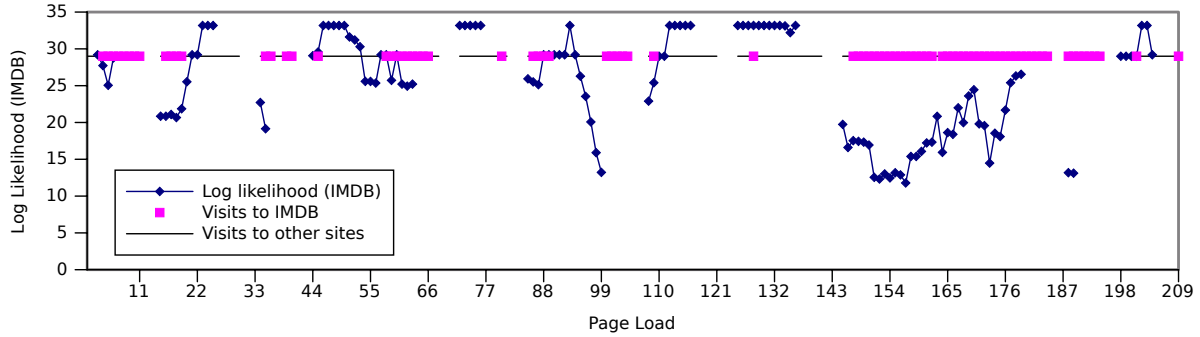
Figure 3.8: Receiver operating curves for the (a) Facebook and (b) IMDB web site classifiers.

Figure 3.8 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 3.6(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 3.9 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,



(a)



(b)

Figure 3.9: 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.

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 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 3.7 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.

Chapter 4

Theoretical Foundations

Before developing an efficient defense system that provides good security, we need to solve the following two problems:

- **How to compare different defenses?** When two defenses are evaluated against the same attack, it can be difficult to compare them, since every defense offers a different trade-off between overhead and security.
- **How efficient can a defense be?** Even if one defense strictly dominates all other defenses in terms of security and efficiency, it is still not clear whether the attack is optimal. We want to know how efficient a defense can be while offering a given level of security.

In this chapter we first develop a model of website fingerprinting attacks and defenses, and derive lower bounds on the bandwidth overhead of any defense that achieves a given level of security. This enables us to compare defenses with different overhead/security trade-offs by comparing how close they are to the optimal trade-off curve. We then analyze the security against any attacker equipped with a multi-feature classifier. We develop the Glove defense system in Chapter 6 based on the model and our analysis. Lastly we show how to derive open-world performance from closed-world experimental results.

4.1 Security vs. Overhead Trade-Off

We focus on understanding the relationship between bandwidth overhead and security guarantees. The overhead required by a fingerprinting defense depends on the set of web sites to be protected – a set of similar websites can be protected with little overhead, a set of dissimilar websites requires more overhead. To derive lower bounds, we consider an

offline version of the website fingerprinting defense problem, i.e. the defense system knows, in advance, the set of websites that the user may visit and the packet traces that each website may generate. We develop an efficient dynamic program to compute a lower bound on the bandwidth overhead of any fingerprinting defense scheme in the closed-world setting.

4.1.1 Definitions

In a website fingerprinting attack, the defender selects a website, w, and uses the defense mechanism to load the website, producing a packet trace, t, that is observed by the attacker. The attacker then attempts to guess w.

Let W be a random variable representing the URL of the website selected by the defender. The probability distribution of W reflects the probability that the defender visits each website. For each website, w, let T_w^D and T_w be the random variables representing the packet trace generated by loading w with and without defense system D, respectively. Packet traces include the time, direction, and content of each packet. Since cryptographic attacks are out of scope for this dissertation, we assume any encryption functions used by the defense scheme are information-theoretically secure. The probability distribution of T_w^D captures variations in network conditions, changes in dynamically-generated web pages, randomness in the browser, and randomness in the defense system. We assume the attacker knows the distribution of W and T_w^D for every w.

In a closed-world setting, the attacker's goal is to infer W from T_W^D. The optimal closed-world attacker, A, upon observing trace t, outputs

$$A(t) = \operatorname*{argmax}_w \Pr[W = w] \Pr\left[T_w^D = t\right]$$

If more than one w attains the maximum, then the attacker chooses randomly among them.

Some privacy applications require good worst-case performance, and some only require good average-case performance. This leads to two security definitions for website fingerprinting defenses:

Definition 1. *A fingerprinting defense D is* ***non-uniformly ϵ-secure*** *for W iff* $\Pr\left[A(T_W^D) = W\right] \leq \epsilon$. *Defense D is* ***uniformly ϵ-secure*** *for W if* $\max_w \Pr\left[A(T_w^D) = w\right] \leq \epsilon$.

These are information-theoretic security definitions – A is the optimal attacker described above. The first definition says that A's average success rate is less than ϵ, but it does not require that every website be difficult to recognize. The second definition requires all websites to be at least ϵ difficult to recognize. All previous papers on website fingerprinting

attacks and defenses have reported average attack success rates in the closed-world model, i.e. they have reported non-uniform security measurements. We will do the same.

To define the bandwidth overhead of a defense system, let $B(t)$ be the total number of bytes transmitted in trace t. We define the **bandwidth ratio** of defense D as

$$\text{BWRatio}_D(W) = \frac{E\left[B\left(T_W^D\right)\right]}{E\left[B\left(T_W\right)\right]}$$

This definition captures the overall bandwidth ratio between a user surfing the web while using defense D and a user visiting the same websites with no defense.

4.1.2 Bandwidth Lower Bounds

In this section we derive an algorithm to compute, given websites $w_1, \ldots, w_n$, a lower bound for the bandwidth that any non-uniformly ϵ-secure fingerprinting defense can use in a closed-world experiment using $w_1, \ldots, w_n$.

To compute a lower bound on bandwidth, we consider an adversary that looks only at the total number of bytes in a packet trace, i.e. an attacker A_S that always guesses

$$A_S(t) = \operatorname*{argmax}_w \Pr\left[B(T_w^D) = B(t)\right]$$

Any defense that is ϵ-secure against an arbitrary attacker must also be at least ϵ-secure against A_S. If we can derive a lower bound on defenses that are ϵ-secure against A_S, that lower bound will apply to any ϵ-secure defense.

We make two simplifying assumptions in order to obtain an efficient algorithm for computing lower bounds. First, we assume that each website has a unique fixed size, s_i. In our closed-world experiments, we found that, for just over half the web pages in our data set, their size had a normalized standard deviation of less than 0.11 across 20 loads, so we do not believe this assumption will significantly impact the results of our analysis. Second, we assume that the defense mechanism does not compress or truncate the website.

We prove the following theorem in Appendix B. Lower Bound Proofs:

Theorem 1. *Suppose ϵn is an integer. Let W be a random variable uniformly distributed over $w_1, \ldots, w_n$, i.e. W represents a closed-world experiment. Suppose D is a defense that is ϵ-non-uniformly-secure against A_S on distribution W. Then there exists a monotonically increasing function f from $S = \{s_1, \ldots, s_n\}$ to itself such that*

- $|f(S)| \leq \epsilon n$.
- $\sum_{i=1}^{n} f(s_i) / \sum_{i=1}^{n} s_i \leq \text{BWRatio}_D(W)$.

Intuitively, f represents a mapping from each website's original size (s_i) to the number of bytes that D transmits when loading website w_i.

This theorem enables us to efficiently compute a lower bound on the overhead of any defense that is ϵ uniformly or non-uniformly secure in a closed-world experiment on $w_1, \ldots, w_n$. To get a lower bound for non-uniformly ϵ-secure defenses, we just need to find a monotonically increasing function $f : S \to S$ that satisfies $|f(S)| \leq \epsilon n$ and minimizes $\sum_{i=1}^{n} f(s_i)$.

Such an f is equivalent to a partition $S_1, \ldots, S_k$ of S satisfying $k \leq \epsilon n$ and minimizing $\sum_{i=1}^{k} |S_i| \max_{s \in S_i} s$. These partitions satisfy a recurrence relation. If $S_1, \ldots, S_k$ is an optimal non-uniformly $\frac{k}{n}$-secure partition, then $S_1, \ldots, S_{k-1}$ is an optimal non-uniformly $\frac{k-1}{n-|S_k|}$-secure partition of $S_1 \cup \cdots \cup S_{k-1}$. Therefore the cost, $C(\frac{k}{n}, n)$, of the optimal f satisfies the recurrence

$$C(\frac{k}{n}, n) = \begin{cases} ns_n & \text{if } k = 1 \\ \min\limits_{1 \leq j \leq n-1} C(\frac{k-1}{n-j}, n-j) + js_n & \text{otherwise.} \end{cases}$$

We can obtain a similar bound for uniformly ϵ-secure deterministic defenses. We say a defense is deterministic if, on each load of website w_i, it always transmits b_i bytes. The following theorem is proven in Appendix B. Lower Bound Proofs.

Theorem 2. *Let W be uniformly distributed over $w_1, \ldots, w_n$, i.e. W represents a closed-world experiment. Suppose D is a deterministic defense that is uniformly ϵ-secure against A_S on distribution W. Then there exists a monotonically increasing function f from $S = \{s_1, \ldots, s_n\}$ to itself such that*

- $\min_i |f^{-1}(s_i)| \geq 1/\epsilon$.
- $\sum_{i=1}^{n} f(s_i) / \sum_{i=1}^{n} s_i \leq \text{BWRatio}_D(W)$.

As with the lower bound on non-uniformly secure defenses, such an f corresponds to a partition $S_1, \ldots, S_k$ of S satisfying $\min_i |S_i| \geq 1/\epsilon$ and minimizing $\sum_{i=1}^{k} |S_i| \max_{s \in S_i} s$. These partitions satisfy a slightly different recurrence. If $S_1, \ldots, S_k$ is is an optimal uniformly ϵ-secure partition of S, then $S_1, \ldots, S_{k-1}$ is an optimal uniformly ϵ-secure partition on $S_1 \cup \cdots \cup S_{k-1}$. Thus the cost, $C(\epsilon, n)$ of the optimal uniformly ϵ-secure partition satisfies the recurrence relation:

$$C'(\epsilon, n) = \begin{cases} \infty & \text{if } n < 1/\epsilon \\ ns_n & \text{if } n \in \left[\frac{1}{\epsilon}, \frac{2}{\epsilon}\right) \\ \min\limits_{1 \leq j \leq \frac{n-1}{\epsilon}} C'(\epsilon, n-j) + js_n & \text{otherwise.} \end{cases}$$

Algorithm 1 shows a dynamic program for computing a lower bound on the bandwidth of any defense that can achieve ϵ non-uniform security in a closed-world experiment on static

Algorithm 1 Algorithm to compute a lower bound on the bandwidth of any offline non-uniformly ϵ secure fingerprinting defense against A_S attackers.

function A_S-MIN-COST(n, ϵ, $\{s_1, \ldots, s_n\}$)
  Array $C[0 \ldots n\epsilon, 0 \ldots n]$
  for $i = 0, \ldots, n\epsilon$ **do**
    $C[i, 0] \leftarrow 0$
  end for
  for $i = 0, \ldots, n$ **do**
    $C[0, i] \leftarrow \infty$
  end for
  for $i = 1 \rightarrow n$ **do**
    for $j = 1 \rightarrow n\epsilon$ **do**
      $C[j, i] = \min_{1 \leq \ell \leq i-1} [(i - \ell)s_i + C[j - 1, \ell]]$
    end for
  end for
  return $C[n\epsilon, n]$
end function

websites with sizes $s_1, \ldots, s_n$ in time $O(n^2\epsilon)$. We use this algorithm to compute the lower bounds reported in Chapter 5.3. The dynamic program for computing uniform security lower bounds is similar.

4.1.3 Security Against Multiple Feature Classifiers

We now analyze the task of defending against attack classifiers that harness multiple features including, but not restricted to trace lengths, trace sizes, packet sizes, and ordering – e.g., the DLSVM attack described above. We will refer to such attackers as MF-attacker. We show that finding the lowest-cost offline defense against MF-attackers is NP-hard, via a reduction from the binary shortest common super-sequence problem. This reduction will also show that the minimum bandwidth required by an offline defense against an MF-attacker is at most twice the bandwidth lower bound computed in the previous section. This result will show that offline defenses can achieve low cost and high security, suggesting a promising avenue for developing a provably secure defense.

Suppose websites $w_1, \ldots, w_n$ are all static and constructed such that loading each site requires performing a fixed, serialized sequence of requests and responses, e.g. each web page contains a javascript program that loads objects one at a time in a fixed order. Let $d_i[j] = 1$ iff the jth byte that must be transmitted to load page w_i is a transmission in the upstream direction.

Loading website w_i via a deterministic defense mechanism produces a fixed trace t_i. Let

z_i be the binary string defined by $z_i[j] = 1$ iff the jth byte of t_i is an upstream byte. Since, for these websites, the defense mechanisms cannot delete or re-order bytes, we must have that d_i is a sub-sequence of z_i.

When the client loads a web site, producing trace t, the attacker can compute the corresponding string, z. In order for the attacker to learn nothing about which web page the client loaded, we must have that, for all i, d_i is a substring of z. Thus the defense system must compute some string, z, that is simultaneously a super-sequence of $d_1, \ldots, d_n$. Minimizing the cost of such a defense is thus equivalent to finding the shortest common super-sequence (SCS) of $d_1, \ldots, d_n$. This problem is NP-hard[25].

However, there is a simple 2-approximation for the binary SCS problem. Let ℓ be the length of the longest string $d_1, \ldots, d_n$. Their SCS must be at least ℓ long, but is at most 2ℓ long, since every binary string of length at most ℓ is a sub-sequence of $(01)^\ell$. Thus for any set of static websites $w_1, \ldots, w_n$, there exists a deterministic offline defense that achieves (uniform or non-uniform) ϵ-security against MF-attackers and incurs bandwidth cost that is at most twice the bandwidth lower bound derived in the previous section.

4.2 From Closed To Open World

Most attack evaluations have used the artificial "closed-world" model, in which the victim selects one of n websites uniformly randomly and the attacker attempts to guess the chosen website based on the observed network traffic. This model has been criticized for being unrealistic because, in a real attack depicted in the "open-world" model, the victim may visit any website in the world[54], potentially making the attacker's task much more difficult. Consequently, some researchers have suggested that website fingerprinting attacks are in fact a paper tiger[54]. However, the two models are connected: our DLSVM attack shows how to bootstrap a closed-world attack into an open-world attack, such that better closed-world performance yields better open-world performance. Thus, although experiments in the closed-world cannot tell us whether an attack or defense will be successful in the real world, we can use closed-world experiments to compare different attacks and defenses.

In this section, we show how to use closed-world experimental results to compute open-world security of defenses and open-world performance of attacks. This makes attack and defense evaluation simpler: researchers need only perform closed-world experiments to predict open-world performance.

In an open-world attack, the defender selects a website, W, according to some probability distribution and generates a trace, T_W^D, corresponding to a visit to that website using some defense, D. The attacker's goal is to determine whether $W = w^*$, where w^* is a particular

website of interest. (It is easy to generalize this definition to situations with multiple websites of interest).

In the open-world setting, the distribution of the random variable W corresponds to the popularity of different websites among the population of users being monitored in the attack. So, for example, if the fingerprinting attacker is a government monitoring citizens Tor usage, then W would be distributed according to the popularity of websites among that nation's Tor users.

Any closed-world attack can be used to construct an open-world attack by selecting websites $w_2, \ldots, w_n$ and building a closed-world classifier, A, on $w^*, w_2, \ldots, w_n$. The open-world classifier is defined as $C(t) = 1$ iff $A(t) = w^*$.

We can compute the false positive rate of this open-world attack as follows. Let $p^* = \Pr[W = w^*]$ and $p_i = \Pr[W = w_i]$ for $i = 2, \ldots, n$. We can obtain estimates for $p^*, p_2, \ldots, p_n$ from public sources, such as the Alexa "Page-Views per Million" database [1]. Let R_n be the average success rate of A in the closed-world, i.e.

$$R_n = \frac{\Pr[A(T^D_{w^*}) = w^*] + \sum_{i=2}^{n} \Pr[A(T^D_{w_i}) = w_i]}{n}$$

Note that R_n is the standard performance metric used in closed-world evaluations. For simplicity, we will assume that $\Pr[A(T^D_{w^*}) = w^*] = R_n$. We also assume that, whenever A misclassifies a trace, there is a $1/n$ chance that it misclassifies the trace as w^*, i.e. that $\Pr[A(T^D_W) = w^* | W \neq w^* \wedge A(T^D_W) \neq W] = 1/n$. Essentially, these two assumptions are equivalent to assuming that w^* is not particularly difficult or easy for A to recognize. With

these assumptions, we can compute C's false-positive rate:

$$\begin{aligned}
\mathrm{FPR}(C) &= \Pr[C(T_W^D) = 1 | W \neq w^*] \\
&= \sum_{w \neq w^*} \frac{\Pr[W = w] \Pr[C(T_w^D) = 1]}{1 - p^*} \\
&= \sum_{w \neq w^*} \frac{\Pr[W = w] \Pr[A(T_w^D) = w^*]}{1 - p^*} \\
&= \sum_{i=2}^{n} \frac{\Pr[W = w_i] \Pr[A(T_{w_i}^D) = w^*]}{1 - p^*} \\
&+ \left(1 - \sum_{i=2}^{n} \Pr[W = w_i]\right) \frac{1}{n(1 - p^*)} \\
&= \frac{1 - R_n}{n(1 - p^*)} \sum_{i=2}^{n} p_i + \frac{1}{n(1 - p^*)} \left(1 - \sum_{i=2}^{n} p_i\right)
\end{aligned}$$

With the same assumptions, the true positive rate of C is

$$\mathrm{TPR}(C) = \Pr[C(T_W^D) = 1 | W = w^*] = R_n$$

The choice of the websites $w_2, \ldots, w_n$ used to build A will affect the performance of C in the open world. The choice of websites affects the false-positive rate in two ways: (1) choosing less popular websites tends to increase the false-positive rate since it decreases $\sum_{i=2}^{n} p_i$, and (2) choosing more similar websites increases the false-positive rate by reducing R_n. The choice of websites affects the true-positive rate only through R_n. Chapter 3.3 shows that the Alexa top 100 websites were about as similar as 100 randomly chosen websites, i.e. that the most popular websites are not particularly similar to each other. Thus it is generally a good strategy to choose $w_2, \ldots, w_n$ to be the most popular websites other than w^*.

Similarly, the number, n, of websites used to build A affects the false-positive rate in two ways: (1) increasing n tends to increase the false positive rate by lowering R_n, and (2) increasing n tends to decrease the false-positive rate since it increases $\sum_{i=2}^{n} p_i$. Increasing n can only decrease the true-positive rate.

Thus we can tune the false-positive and true-positive rates of C by varying n. Small n will have large true- and false-positive rates. Increasing n will reduce both the false- and true-positive rates. By varying n, we can generate the receiver operating curve (ROC) of C.

In the real world, visits to w^* may be rare. In this case, false-positive rate can be a misleading metric. A classifier with a low false-positive rate may still be useless if true

positives are so rare that they are overwhelmed by false positives. Therefore, we also report true-discovery rates for the open-world attack and defense evaluations in this dissertation. Given an open-world classifier, C, its true-discovery rate is defined as

$$\mathrm{TDR}(C) = \Pr[W = w^* | C(T_W^D) = 1].$$

Intuitively, the true-discovery rate is the fraction of alarms that are true alarms. The true-discovery rate can be computed from the false-positive and true-positive rates as follows:

$$\begin{aligned}\mathrm{TDR}(C) &= \frac{\Pr[W = w^*]\,\mathrm{TPR}(C)}{\Pr[W = w^*]\,\mathrm{TPR}(C) + \Pr[W \neq w^*]\,\mathrm{FPR}(C)} \\ &= \frac{p^* R_n}{p^* R_n + \frac{1-R_n}{n}\sum_{i=2}^{n} p_i + (1 - \sum_{i=2}^{n} p_i)\frac{1}{n}}\end{aligned}$$

Chapter 5

Congestion-Sensitive BuFLO

5.1 Design

Dyer, et al., described BuFLO, a hypothetical defense scheme that hides all information about a website, except possibly its size, and performed a simulation-based evaluation that found that, although BuFLO is able to offer good security, it incurs a high cost to do so.

In this section, we describe Congestion-Sensitive BuFLO (CS-BuFLO), an extension to BuFLO that includes numerous security and efficiency improvements. CS-BuFLO represents a new approach to the design of fingerprinting defenses. Most previously-proposed defenses were designed in response to known attacks, and therefore took a black-listing approach to information leaks, i.e. they tried to hide specific features, such as packet sizes. In designing CS-BuFLO, we take a white-listing approach – we start with a design that hides all traffic features, and iteratively refine the design to reveal certain traffic features that enable us to achieve significant performance improvements without significantly harming security.

5.1.1 Review of BuFLO

The Buffered Fixed-Length Obfuscator (BuFLO) of Dyer, et al., transmits a packet of size d bytes every ρ milliseconds, and continues doing so for at least τ milliseconds. If $b < d$ bytes of application data are available when a packet is to be sent, then the packet is padded with $d - b$ extra bytes of junk. The protocol assumes that the junk bytes are marked so that the receiver can discard them. If the website does not finish loading within τ milliseconds, then BuFLO continues transmitting until the website finishes loading and then stops immediately. Dyer, et al., did not specify how BuFLO detects when the website has finished loading. They also did not specify how BuFLO handles bidirectional communication – presumably independent BuFLO instances are run at each end-point.

BuFLO effectively hides everything about the website, except possibly its size, but has several shortcomings:

- It either completely hides the size of the website or completely reveals it ($\pm d$ bytes). Thus it does not provide the same level of security to all websites.
- BuFLO has large overheads for small websites. Thus its overhead is also unevenly distributed.
- BuFLO is not TCP-friendly. In fact, it is the epitome of a bad network citizen.
- BuFLO does not adapt when the user is visiting fast or slow websites. It wastes bandwidth when loading slow sites, and causes large latency when loading fast websites.
- BuFLO must be tuned to each user's network connection. If the BuFLO bandwidth, $\frac{1000d}{\rho}$ B/s, exceeds the user's connection speed, then BuFLO will incur additional delay without improving security.
- Past research by Fu, et al., showed that transmitting at fixed intervals can reveal load information at the sender, which an attacker can use to infer partial information about the data being transmitted[24].

Dyer, et al., proposed BuFLO as a straw-man defense system, so it is understandable that they did not bother addressing these problems. However, we show below that several of these problems have common solutions, e.g. we can simultaneously improve overhead and TCP-friendliness, simultaneously make security and overhead more uniform, etc. Thus, as our evaluation will show, CS-BuFLO may be a practical and efficient defense for users requiring a high level of security.

Further, as noted by its authors, BuFLO's simulation based results "reflect an ideal implementation that assumes the feasibility of implementing fixed packet timing intervals. This is at the very least difficult and clearly impossible for certain values of ρ. Simulation also ignores the complexities of cross-layer communication in the network stack" [20]. As a result, it remains unclear how well the defense performs in the real world.

5.1.2 Overview of Congestion-Sensitive BuFLO

Algorithm 2 shows the main loop of the CS-BuFLO server. The client loop is similar, except for the few differences discussed throughout this section. Similarly to BuFLO, CS-BuFLO delivers fixed-size chunks of data at semi-regular intervals. CS-BuFLO randomizes the timing of network writes in order to counter the attack of Fu, et al.[24], but it maintains

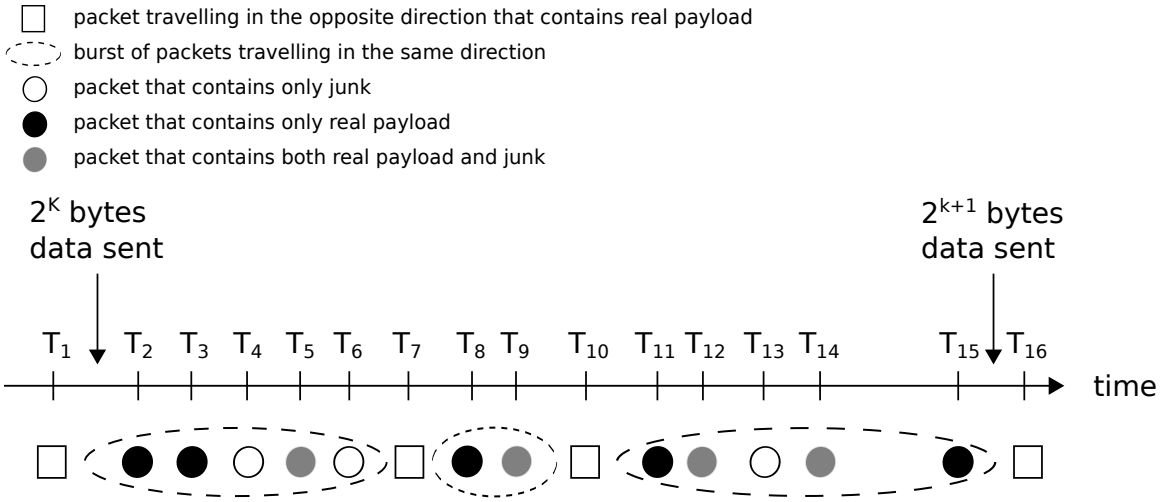


Figure 5.1: Rate adaptation in CS-BuFLO. ρ^* is updated based on the packets transmitted to the other end between T_2 and T_{15}. Time intervals between two consecutive packets are stored in an array $Intervals[]$. The two packets under consideration both contain some real payload data and they belong to the same burst. i.e. $Intervals = [T_3 - T_2, T_5 - T_3, T_9 - T_8, T_{12} - T_{11}, T_{14} - T_{12}, T_{15} - T_{14}]$ and $\rho^* = 2^{\lfloor \log_2 Median(Intervals[]) \rfloor}$.

a target average inter-packet time, ρ^*. CS-BuFLO periodically updates ρ^* to match its bandwidth to the rate of the sender (Chapter 5.1.3). Since updating ρ^* based on the sender's rate reveals information about the sender, CS-BuFLO performs these updates infrequently. CS-BuFLO uses TCP to be congestion friendly, and uses feedback from the TCP stack in order to reduce the amount of junk data it needs to send (Chapter 5.1.4). Also like BuFLO, CS-BuFLO transmits extra junk data after the website has finished loading in order to hide the total size of the website. However, CS-BuFLO uses a scale-independent padding scheme (Chapter 5.1.5) and monitors the state of the page loading process to avoid some unnecessary overheads (Chapter 5.1.6).

5.1.3 Rate Adaptation

CS-BuFLO adapts its transmission rate to match the rate of the sender. This reduces wasted bandwidth when proxying slow senders, and it reduces latency when proxying fast senders. However, adapting CS-BuFLO's transmission rate to match the sender's reveals information about the sender, and therefore may harm security.

As shown in Figure 5.1, CS-BuFLO takes several steps to limit the information that is leaked through rate adaptation. First, it only adapts after transmitting 2^k bytes, for some integer k. Thus, during a session in which CS-BuFLO transmits n bytes, CS-BuFLO will perform $\log_2 n$ rate adjustments, limiting the information leaked from these adjustments.

Algorithm 2 The main loop of the Congestion-Sensitive BuFLO server.

```
function CSBUFLO-SERVER(s)
    while true do
        (m, ρ) = READ-MESSAGE(ρ)
        if m is application data from website then
            output-buff ← output-buff ‖ data
            real-bytes ← real-bytes + LENGTH(m)
            last-site-response-time ← CURRENT-TIME
        else if m is application data from client then
            send m to the website
            ρ-stats ← ρ-stats ‖ ⊥
            onLoadEvent ← 0, padding-done ← 0
        else if m is onLoad message then
            onLoadEvent ← 1
        else if m is padding-done message then
            padding-done ← 1
        else if m is a time-out then
            output-buff-bytes ← LENGTH(output-buff)
            (output-buff, j) ← CS-SEND(s, output-buff)
            junk-bytes ← junk-bytes + j
            real-bytes-sent ← real-bytes-sent + output-buff-bytes - LENGTH(output-buf) - j
            if real-bytes-sent ≥ PACKET-SIZE then
                real-bytes-sent ← real-bytes-sent - PACKET-SIZE
                ρ-stats ← ρ-stats ‖ CURRENT-TIME
            end if
        end if
        if DONE-XMITTING then
            reset all variables
        else                                   ▷ ρ* : Average time between sends to client
            if ρ* = ∞ then
                ρ* ← INITIAL-RHO
            else if CROSSED-THRESHOLD(real-bytes + junk-bytes) then
                ρ* ← RHO-ESTIMATOR(ρ-stats,ρ*)
                ρ-stats ← ∅
            end if

            if m is a time-out then
                ρ ← random number in [0, 2ρ*]
            end if
        end if
    end while
end function
```

Algorithm 3 Algorithm for estimating new value of ρ^* based on past network performance.

function RHO-ESTIMATOR(ρ-*stats*, ρ^*)
  $I \leftarrow [\rho\text{-}stats_{i+1} - \rho\text{-}stats_i \mid \rho\text{-}stats_i \neq \perp \wedge \rho\text{-}stats_{i+1} \neq \perp]$
  if I is empty list **then**
    return ρ^*
  else
    return $2^{\lfloor \log_2 \text{MEDIAN}(I) \rfloor}$
  end if
end function

This choice also allows CS-BuFLO to adapt more quickly during the beginning of a session, when the sender is likely to be performing a TCP slow start. During this phase, CS-BuFLO is able to ramp up its transmission rate just as quickly as the sender can.

CS-BuFLO further limits information leakage by using a robust statistic to update ρ^*. Between adjustments, it collects estimates of the sender's instantaneous bandwidth. It then sets ρ^* so as to match the sender's median instantaneous bandwidth. Median is a robust statistic, meaning that the new ρ^* value will not be strongly influenced by bandwidth bursts and lulls, and hence ρ^* will not reveal much about the sender's transmission pattern.

Note that the estimator only collects measurements during uninterrupted bursts from the sender. This ensures that the bandwidth measurements do not include delays caused by dependencies between requests and responses.

For example, if the estimator sees a packet p_1 from the website, then a packet p_2 from the client, and then another packet p_3 from the website, it may be the case that p_3 is a response to p_2. In this case, the time between p_1 and p_3 is constrained by the round trip time, not the website's bandwidth.

Finally, CS-BuFLO rounds all ρ^* values up to a power of two. This further hides information about the sender's true rate, and gives the sender room to increase it's transmission rate, e.g. during slow start.

5.1.4 Congestion-Sensitivity

There's a trivial way to make BuFLO congestion sensitive and TCP friendly: run the protocol over TCP. However, this simple approach misses an opportunity for increasing efficiency: when the network is congested, BuFLO does not need to insert junk data to fill the output buffer.

Algorithm 4 shows our method for taking advantage of congestion to reduce the amount of junk data sent by CS-BuFLO. Note first that CS-SEND always writes exactly d bytes to the TCP socket. Since the amount of data presented to the TCP socket is always the same,

Algorithm 4 Algorithm for sending data and using feedback from TCP. Socket s should be configured with `O_NONBLOCK`.

```
function CS-SEND(s, output-buff)
    n ←LENGTH(output-buff)
    j ← 0
    if n < PACKET-SIZE then
        j ← PACKET-SIZE − n
        output-buff ← output-buff ‖ j
    end if
    r ← write(s, output-buff, PACKET-SIZE)
    if r ≥ n then                          ▷ Optional: reclaim unsent junk
        output-buff ← empty buffer
        j ← r − n
    else
        remove last j bytes from output-buff
        remove first r bytes from output-buff
        j ← 0
    end if
    return (output-buff, j)
end function
```

this algorithm reveals no information about the timing or size of application-data packets from the website that have arrived at the CS-BuFLO proxy.

This algorithm takes advantage of congestion to reduce the amount of junk data it sends. To see why, imagine the TCP connection to the client stalls for an extended period of time. Eventually, the kernel's TCP send queue for socket s will fill up, and the call to `write` will return 0. From then until the TCP congestion clears up, CS-BuFLO calls to CS-SEND will not append any further junk data to B.

5.1.5 Stream Padding

CS-BuFLO hides the total size of real data transmitted by continuing to transmit extra junk data after the browser and web server have stopped transmitting.

Table 5.1 shows two related padding schemes we experimented with in CS-BuFLO. Both schemes introduce at most a constant factor of additional cost, but reveal at most a logarithmic amount of information about the size of the website. The first scheme, which we call *payload* padding, continues transmitting until the total amount of transmitted data $(R + J)$ is a multiple of $2^{\lceil \log_2 R \rceil}$. This padding scheme will transmit at most $2^{\lceil \log_2 R \rceil}$ additional bytes, so it increases the cost by at most a factor of 2, but it reveals only $\log_2 R$.

The second scheme, which we call *total* padding, continues transmitting until $R + J$ is a

Padding Schemes	Payload Sent Before Padding	Junk Sent Before Padding	Total Bytes Sent After Padding
payload padding	R	J	$c2^{\lceil \log_2 R \rceil}$
total padding	R	J	$2^{\lceil \log_2 (R+J) \rceil}$

Table 5.1: Two different padding schemes for CS-BuFLO.

Algorithm 5 Definition of the DONE-XMITTING function.

```
function DONE-XMITTING
    return LENGTH(output-buff) ← 0 ∧CHANNEL-IDLE(onLoadEvent,last-site-response-time)∧
(padding-done ∨ CROSSED-THRESHOLD(real-bytes + junk-bytes))
end function

function CHANNEL-IDLE(onLoadEvent, last-site-response-time)
    return onLoadEvent ∨ (last-site-response-time + QUIET-TIME < CURRENT-TIME)
end function

function CROSSED-THRESHOLD(x)
    return ⌊log2(x − PACKET-SIZE)⌋ < ⌊log2 x⌋
end function
```

power of 2. This also increases the cost by at most a factor of 2 and reveals, in the worst case, $\log_2 R$, but it will in practice hide more information about R than payload padding.

Note that the CS-BuFLO server and the CS-BuFLO client do not have to use the same stream padding scheme. Thus, there are four possible padding configurations, which we denote CPSP (client payload, server payload), CPST (client payload, server total), CTSP (client total, server payload) and CTST (client total, server total).

In order to determine when to stop padding, the CS-BuFLO server must know when the website has finished transmitting. Congestion-Sensitive BuFLO uses two mechanisms to recognize that the page has finished loading. First, the CS-BuFLO client proxy monitors for the browser's onLoad event. The CS-BuFLO client notifies the CS-BuFLO server when it receives the onLoad event from the browser. Once the CS-BuFLO server receives the onLoad message from the client, it considers the web server to be idle (see Algorithm 5) and will stop transmitting as soon as it adds sufficient stream padding and empties its transmit buffer. As a backup mechanism, the CS-BuFLO server considers the website idle if QUIET-TIME seconds pass without receiving new data from the website. We used a QUIET-TIME of 2 seconds in our prototype implementation.

5.1.6 Early Termination

As described above, the CS-BuFLO server is likely to finish each page load by sending a relatively long tail of pure junk packets. This tail can be a significant source of overhead and, somewhat surprisingly, may not provide much additional security.

Our initial investigations revealed that the long tail served two purposes which could also be served through other, more efficient means. As mentioned above, the long tail helps hide the total size of the website. However, the interior padding performed by CS-SEND also obscures the total size of the website. Our evaluation in Chapter 5.3 investigates the security impact of additional stream padding.

In the specific context of web browsing, the long tail also hides the size of the last object sent from the web server to the client. The attacker can infer some information about the size of this object by measuring the amount of data the CS-BuFLO server sends to the CS-BuFLO client after the CS-BuFLO client stops transmitting to the CS-BuFLO server. However, this information can also be hidden by having the CS-BuFLO client continue to send junk packets to the CS-BuFLO server, i.e. more aggressive stream padding from the CS-BuFLO client may obviate the need for aggressive padding at the CS-BuFLO server.

Based on these ideas, we implemented an *early termination* feature in our CS-BuFLO prototype. The CS-BuFLO client notifies the CS-BuFLO server that it is done padding. After receiving this message, the CS-BuFLO server will stop transmitting as soon as the web server becomes idle and its buffers are empty.

Figure 5.2 illustrates how the padding scheme used by the client and server can interact, including the impact of early termination. Additional client padding can hide the size of the last HTTP object, and early termination can avoid unnecessary padding. Our evaluation investigates the overhead/security trade-offs between different padding regimes at the client and server, and how they interact with early termination.

5.1.7 Packet Sizes

Sending fixed-length packets hides packet size information from the attacker. Although any fixed length should work, it is important to choose a packet length that maximizes performance. Since we may transmit pure dummy packets during the transmission, larger packets tend to cause higher bandwidth overhead, and on the other hand, smaller packets may not make full use of the link between the client and server, thus increase the loading time.

Preliminary investigations revealed that over 95.7% of all upstream packet transmissions are under 600 bytes, therefore, this was used as the standard packet size in our experiments.

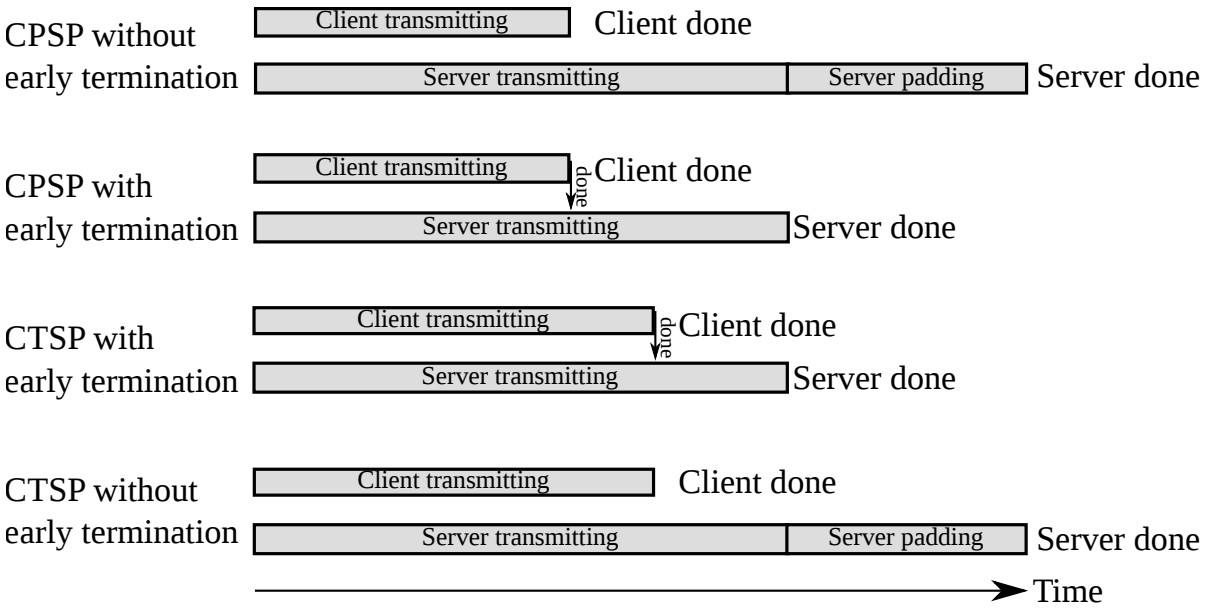


Figure 5.2: The interaction between client and server padding schemes and early termination. More padding at the client can help hide the size of the last object sent from the server to the client. Early termination can avoid unnecessary padding at the end of a page load.

5.2 Prototype Implementation

We modified OpenSSH5.9p1 to implement Algorithm 2. However, the optional junk recovery algorithm described in Algorithm 4 was not implemented.

The SSH client was also modified to accept a new SOCKS proxy command code, *onLoadCmd*. This command was used to communicate to the server when to stop padding (as described in Chapter 5.1.5). A Firefox plugin, *OnloadNotify*, that, upon detecting the page *onLoad* event, connects to the SSH client's SOCKS port and issues the *onLoadCmd*, was also developed.

In addition, the following OpenSSH message types were used:

1. The OpenSSH message type `SSH_MSG_IGNORE`, which means all payload in a packet of this type can be ignored, was used to insert junk data whenever needed.

2. The `SSH_MSG_NOTIFY_ONLOAD` message was created to be used by the client to communicate reception of *onLoadCmd* from the browser, to the server. Upon receiving this message from the client, the CS-BuFLO server stops transmitting as soon as it empties its buffer and adds sufficient stream padding.

3. The `SSH_MSG_NOTIFY_PADDINGDONE` message was created to implement the early termination feature of CS-BuFLO. Upon receiving this message from the client, the CS-

BuFLO server stops transmitting as soon as the web server becomes idle and its buffers are empty.

All the above messages were buffered and transmitted just like other messages in Algorithm 2, i.e. using CS-SEND, therefore an attacker is unable distinguish these messages from other traffic.

5.3 Congestion-Sensitive BuFLO Evaluation

We investigated several questions during our evaluation of Congestion-Sensitive BuFLO:

- How do the different stream padding schemes affect performance and security of CS-BuFLO? What is the effect of adding early termination to the protocol?
- How does CS-BuFLO's security and overhead compare to Tor's, and how do they both compare to the theoretical minimums derived in Chapter 4.1.2?
- Can we use the theoretical lower bounds to enable us to compare defenses that have different overhead/security trade-offs?

5.3.1 Experimental Setup

For our main experiments, we collected traffic from the Alexa top 200 functioning, non-redirecting web pages using four different defenses: plain SSH, Tor, CS-BuFLO with the CTSP padding and early termination, and CS-BuFLO with CPSP padding and early termination. We also collected several smaller data sets using other configurations of CS-BuFLO, but these are only used in the padding scheme evaluations (Table 5.2).

We constructed a list of the Alexa top 200 functioning, non-redirecting, unique pages, as follows. We removed web pages that failed to load in Firefox (without Tor or any other proxy). We replaced URLs that redirected the browser to another URL with their redirect target. Some websites display different languages and contents depending on where the page is loaded, e.g. *www.google.com* and *www.google.de*. We kept only one URL for this type of website, i.e. we only had *www.google.com* in our set. Our data set consisted of Alexa's 200 highest-ranked pages that met these criteria.

We collected 20 traces of each URL, clearing the browser cache between each page load. We collected traces from each web page in a round-robin fashion. As a result, each load of the same URL occurred about 5 hours apart.

Measuring the precise latency of a fingerprinting defense scheme poses a challenge: we can easily measure the time it takes to load a page using the defense, but we cannot infer

the exact time it *would have taken* to load the page without the defense. Therefore, every time we loaded a page using a defense, we immediately loaded it again using SSH to get an estimate of the time it would have taken to load the page without the defense in place. We then compute latency ratios the same way we compute bandwidth ratios, i.e. if $L(t)$ is the total duration of a packet trace, the latency ratio of a defense scheme is

$$\frac{E\left[L(T_W^D)\right]}{E\left[L(T_W)\right]}$$

We collected network traffic using several different computers with slightly different versions of Ubuntu Linux – ranging from 9.10 to 11.10. We used Firefox 3.6.23-3.6.24 and Tor 0.2.1.30 with polipo HTTP Proxy. All Firefox plugins were disabled during data collection, except when collecting CS-BuFLO traffic, where we enabled the *OnloadNotify* plugin. Three of the computers had 2.8GHz Intel Pentium CPUs and 2GB of RAM, one computer had a 2.4GHz Intel Core 2 Duo CPU with 2GB of RAM. We scripted Firefox using Ruby and captured packets using tshark, the command-line version of wireshark. For the SSH experiments, we used OpenSSH5.3p1. Our Tor clients used the default configuration. SSH tunnels passed between two machines on the same local network.

We measured the security of each defense by using the three best traffic analysis attacks in the literature: VNG++ [20], the Panchenko SVM [50], and DLSVM described in Chapter 3. We ran each of the above classifiers against the traces generated by each defense using stratified 10-fold cross validation.

5.3.2 Results

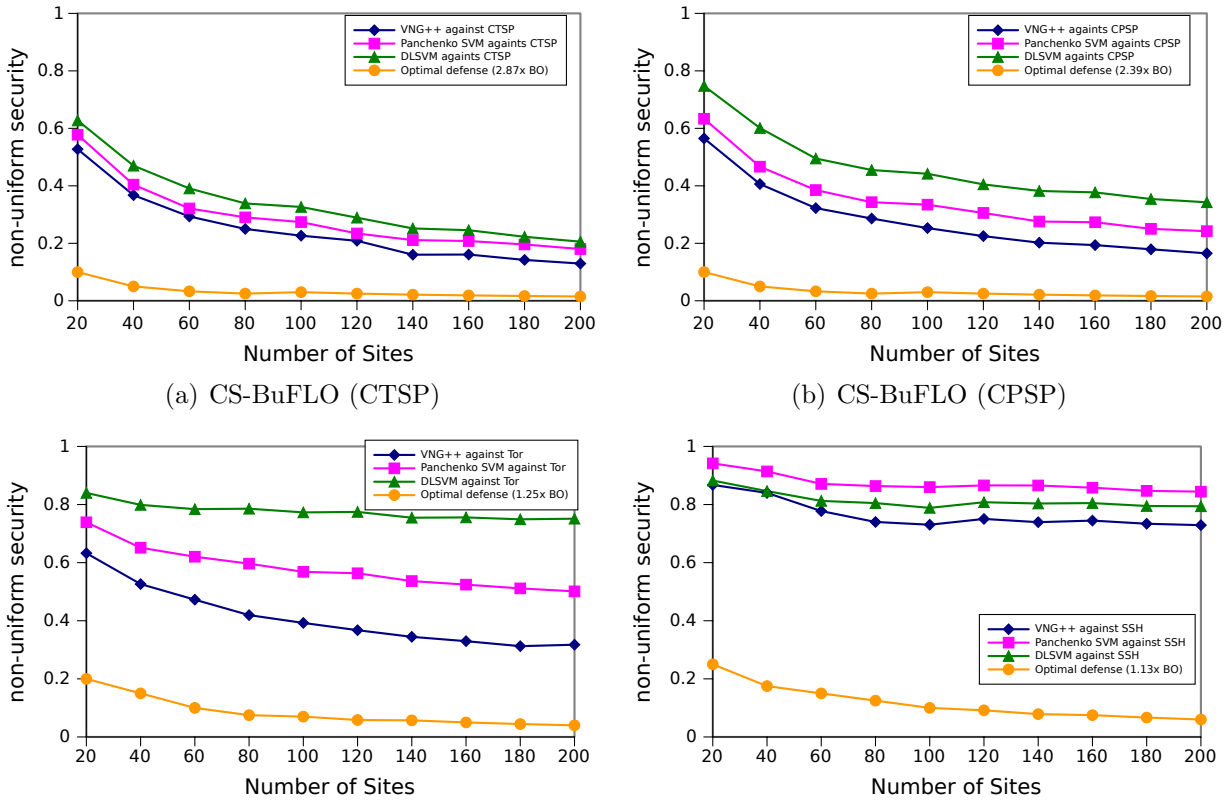
Padding	Early Termination	Bandwidth Cost	Latency	VNG++ Accuracy
CTSP	Yes	3.59	3.91	29.0%
CTSP	No	3.73	3.51	29.6%
CPSP	Yes	2.60	2.87	34.2%
CPSP	No	3.42	3.52	36.0%

Table 5.2: Security and performance of Congestion-Sensitive BuFLO variants. VNG++ success rate is the probability that the attack was able to correctly guess which of 50 web pages the user was visiting.

Padding Schemes. Table 5.2 shows the bandwidth ratio, latency ratio, and security (estimated using the VNG++ attack) of four different versions of CS-BuFLO on a data set

of 50 websites. Note that early termination does not appear to affect security, although it can significantly reduce overhead in some configurations. All other experiments in this dissertation use early termination. The client padding scheme, on the other hand, appears to control a trade-off between overhead and security. Therefore we report the rest of our results for both CPSP and CTSP padding.

Security Comparison. Figure 5.3 shows the level of security various defense schemes provide against three different attacks, as the number of web pages the attacker needs to distinguish increases. Note that the CS-BuFLO schemes have significantly better security than Tor and SSH. For each defense scheme, we compute its average bandwidth cost, BO, and plot the lower bound on security that can be achieved within that cost, using the algorithm from Chapter 4.1.2.



(a) CS-BuFLO (CTSP) (b) CS-BuFLO (CPSP)

(c) Tor (d) SSH

Figure 5.3: Security of CS-BuFLO, Tor, and SSH compared to the optimal defense from Chapter 4.1.2, as a function of the number of possible web pages.

Bandwidth Cost. Figure 5.4 plots the costs of SSH, Tor, and CS-BuFLO with CTSP and CPSP padding. SSH has almost no overhead, and Tor's overhead is about 25% on average. CS-BuFLO with CPSP has an average overhead of 129%, CTSP has average overhead 180%. Thus CS-BuFLO's improved security does come at a price.

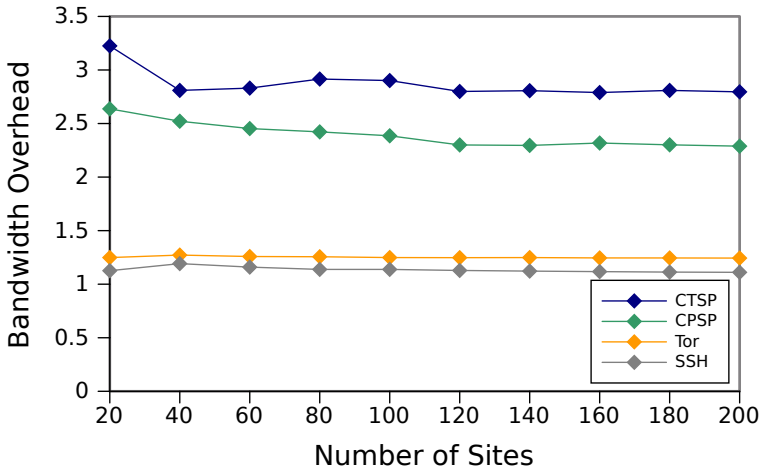


Figure 5.4: Bandwidth costs of various defense schemes as a function of the number of possible web pages.

Theoretical Bounds. Figure 5.5 evaluates CS-BuFLO, Tor, SSH, and BuFLO against the theoretical lower bounds developed in Chapter 4.1.2.

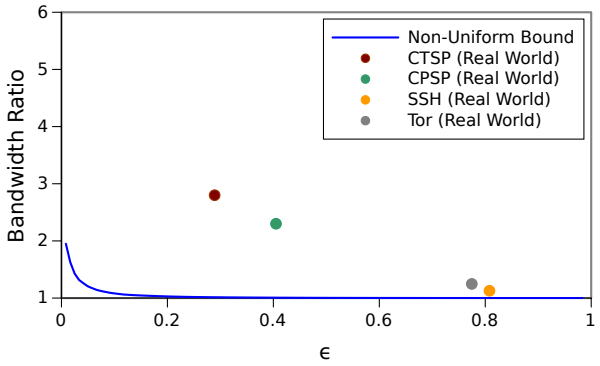
Figure 5.5(a) shows the results of our empirical evaluation of CS-BuFLO, Tor, and SSH on $n = 120$ sites and using the DLSVM attack to estimate security. We limit to 120 sites to make it easier to compare with the BuFLO results reported by Dyer, et al., and which use 128 sites. There is a significant gap between the bandwidth of CS-BuFLO and the lower bound. However, as can be seen in Figure 5.5(c), CS-BuFLO in CTSP mode is over 6× closer to the trade-off lower bound than Tor for 200 sites, and is the most efficient scheme across all sizes we measured.

Figure 5.5(b) presents the results of our empirical evaluation of CS-BuFLO, Tor, and SSH on $n = 120$ websites, using the Panchenko attack to estimate security. We also present Dyer's reported results from their experiments with BuFLO on 128 sites, also using the Panchenko attack. Note that, since Dyer used 128 sites to evaluate BuFLO, this slightly over-estimates BuFLO's security compared to the other schemes plotted in the figure. Also, recall that Dyer's experiments with BuFLO were all based on simulation.

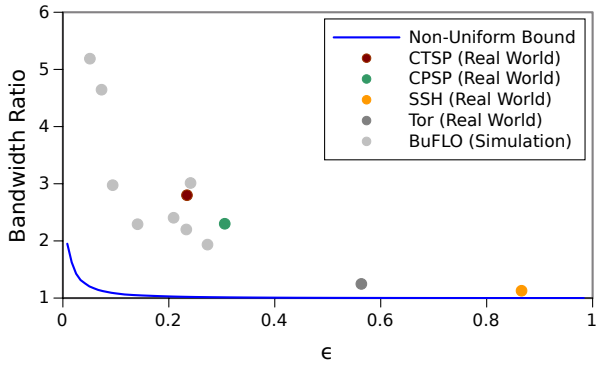
Despite the differences in experimental methodology, we can see that CS-BuFLO offers performance in the same general range as the BuFLO configurations from Dyer's paper, but has slightly worse security in our experiments.

Figure 5.5(d) shows that, based on our experiments and the simulation results of Dyer, et al., all but one BuFLO configuration get closer to the trade-off lower bound curve than CS-BuFLO, Tor, and SSH (SSH is omitted from the graph because its ratio to the lower bound was never less than 400). This figure also highlights a difference between the DLSVM and Panchenko attacks. In the DLSVM results shown in Figure 5.5(c), Tor and SSH diverge

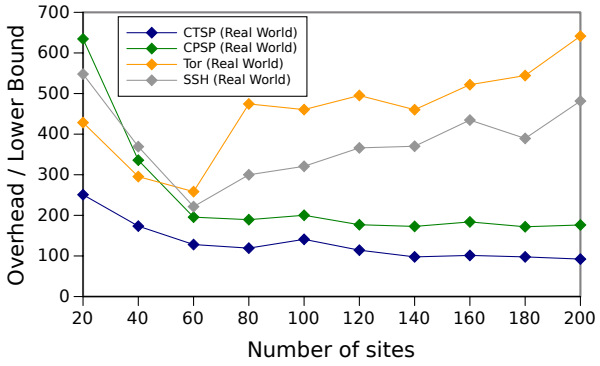
from CS-BuFLO. In the Panchenko results in Figure 5.5(d), Tor and CS-BuFLO appear to be equally close to the lower bound.



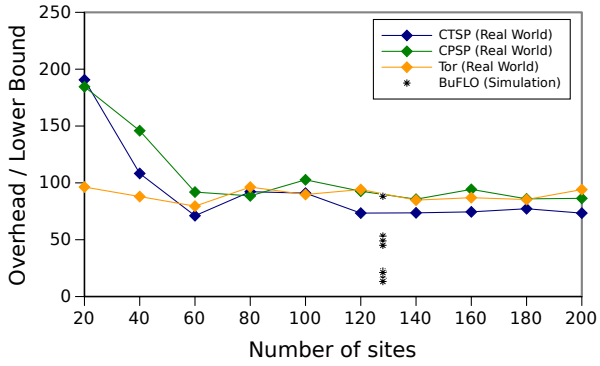
(a) DLSVM, $n = 120$



(b) Panchenko, $n = 120$



(c) DLSVM



(d) Panchenko

Figure 5.5: Non-uniform lower bounds on bandwidth ratio, as a function of the security parameter, ϵ, and specific trade-off points of the systems evaluated. The BuFLO results are taken from Dyer, et al. [20], and therefore use $n = 128$. SSH is omitted from Figure 5.5(d) because its ratio to the lower bound was always greater than 400.

Chapter 6

Glove

Previous work on fingerprinting defenses have evaluated security of defenses by testing them against known attacks. Unfortunately, this approach can only show that a defense is ineffective, but cannot prove that a defense is secure. In this chapter, we present Glove, a defense scheme that is designed to defeat an ideal attacker, who can not distinguish two websites if and only if they generate the same sequence of network traffic observation. Thus the security provided by our defense is attack-independent therefore, we call Glove a provably secure defense, because it gives an upper bound on the success rate of any attack, regardless of its attacking method.

6.1 Design

The gap between the performance of existing defenses and the theoretical lower bounds derived in Chapter 4.1.2 suggests that, if a defense remembers information about websites seen in the past, it may be able to insert the cover traffic in a smart way that incurs little bandwidth overhead. In Chapter 4.1.3 we analyzed how to defend against MF-attackers using the shortest common super-sequence(SCS). We build Glove based on these insights.

Glove consists of an offline *training* phase and an online *defending* phase. During the training phase, Glove collects traces of web pages, clusters the pages by their network-level features, and computes, for each cluster, a transcript of packet sizes and timings. During the defending phase, Glove replays the transcript whenever a user loads one of the pages in that cluster. Glove can also be viewed as simply optimizing for the common case in CS-BuFLO. Instead of sending packets between the proxy end-points at fixed schedules, Glove optimizes this approach by using prior knowledge about popular websites to select a packet schedule that uses less bandwidth but still hides the identity of the website. For websites for which Glove does not have prior knowledge, it falls back to CS-BuFLO.

6.1.1 Security Guarantee

The network traffic generated by loading a web page consists of a sequence of packets, which we call a trace. Packets in a trace can be divided into two categories: request packets transmitted from a client to a web server, and response packets directed to the client. From a different point of view, we can say the contents of a web page are covered by its trace. We also define *super-trace* as follows:

Definition 2 (Super-Trace)**.** *We say that a trace S is the super-trace of traces $t_1, \ldots, t_n$ iff each trace t_i may be transformed into trace S by some sequence of the following actions:*

- ***Inserting:*** *Inserting request or response packets.*
- ***Merging:*** *Merging consecutive requests or responses.*
- ***Splitting:*** *Splitting a packet into a set of smaller sized packets such that the sum of their sizes is maintained.*
- ***Delaying:*** *Increasing delays between a response packet and its succeeding request packet, and vice-versa.*

Clearly, from Definition 2 we can see that, trace t_i of web page w_i can be replaced by *super-trace* S (at the cost of additional bandwidth and latency overhead). This is because S has enough data packets to cover the contents in w_i, while maintaining the temporal dependencies between requests and responses in t_i. To protect n websites $w_1, \ldots, w_n$, Glove does two things: First, it divides the traces of these websites into k clusters based on rules we will describe later. Second, for all the traces within a cluster, Glove computes a single *super-trace*.

Observe that Glove plays the same trace S_c, i.e. the transcript, whenever a web page in cluster c is loaded, thus generating the same observation to an attacker. Glove therefore meets the information theoretic definitions of security described in Definition 1. For example, let C be the smallest cluster among all k clusters. Because loading all web pages in C yield the same observation to an attacker A, the probability that A can correctly guess which web page is loaded is $\frac{1}{|C|}$. Since C is the smallest cluster, Glove is a *uniformly ϵ-secure* defense, where $\epsilon = \frac{1}{|C|}$ here. Similarly, Glove can be tuned to achieve *non-uniformly ϵ security*.

6.1.2 Clustering Web Pages

Because Glove only plays the super-trace during the defending phase, the bandwidth overhead of the defense depends on how we cluster the web page traces and compute the

super-trace. Intuitively, if we put traces that have similar sizes, packet orderings and timings into the same cluster, it is easier to compute a super-trace with low overhead for the cluster.

To find traces that have similar packet orderings and timings, we can treat a trace as a time series, i.e. a byte sequence with time stamps. The time stamp of each byte is the time at which the byte is transmitted.*Dynamic Time Warping (DTW)* is an algorithm to measure the similarity between two time series, thus it can be used to compare two web page traces.

We employ *k-medoids* clustering on the *Dynamic Time Warping (DTW)* based distance matrix computed on input web page traces. *k-medoids* is preferred for clustering as it was designed with the idea of enabling custom distance metrics between points (unlike k-means and other clustering methods which assume a Euclidean space).

Finding Representative Traces: In reality, not all web pages are static and the network conditions change over time, so the trace varies each time a web page is loaded. Before clustering web pages, we need to choose a "representative" trace for each web page, i.e., the trace that is likely to be most easily transformed into other traces generated by the same page. To do this, we load a web page n times and record n traces. We then compute pair-wise Damerau-Levenshtein edit distances [48] among them, and rank the traces based on their average distances to others — the trace with the minimum average distance is ranked $1st$, and is chosen as the representative trace for the web page.

k-medoids Clustering: Since a trace can be viewed as a time-series, we compute the Dynamic Time Warping (DTW) [9] distances between every pair of representative traces. Once this pairwise distance matrix is computed, we use the k-medoids [37] algorithm to group similar web pages into a pre-determined number of clusters. The number of clusters determines the security and overhead of the Glove defense.

Simulations revealed that the clusters generated by DTW had lower overhead super-traces than other distance metrics. This is likely because DTW implicitly takes into account the time between packets while computing edit distances – an important factor when considering that super-traces need to maintain the inter-packet time dependencies between requests and responses in their constituent traces.

In our implementation of k-medoids, the cost of a cluster configuration was the lower-bound on bandwidth overhead which can be computed as: $\sum_i^k |c_i|(\max(req_{j \in c_i}) + \max(res_{j \in c_i}))$. Here, $c_1, \ldots, c_k$ are the k clusters and req_j, res_j denote the number of request and response bytes of the j^{th} site. The idea is that any super-trace of a group of traces must contain at least as many request and response bytes as its constituent traces with the most number of request and response bytes.

6.1.3 Computing Super-traces

The *super-trace* of a cluster is a single trace that covers all traces contained in the cluster. If all web pages are static, a defense that *plays* this super-trace while loading any web page within the cluster, effectively hides all information about the page being loaded (except the cluster it belongs to). However, since most web pages are dynamic, we compute a super-trace that aims to conservatively cover a large (tunable) percentage of all traces that one of its constituent web pages might generate. To do this, we use the heuristic demonstrated in Algorithm 6 to approximate the minimum bandwidth super-trace for each cluster. The following notation is used:

- Minimum site coverage (μ_{min}): This parameter determines the minimum number of traces of each web page to be covered by the super-trace. The parameter μ denotes the average coverage of all pages. Larger μ_{min} values provide more resistance to the dynamicity of web pages, often at the cost of larger overheads.

- T is the set of input traces that the super-trace is computed over. This is initially $\emptyset$. ST denotes the currently computed super-trace. R_i is the set containing all the recorded traces of site i. *covmin* is the index of the site which is least covered by the current ST. This value may be initialized randomly.

- F denotes the current frontier packet of each trace in T and len_i denotes the number of packets in the i^{th} trace in T.

- Bandwidth-Latency tuner (τ): The time at which a newly added packet is to be sent is set to be the time of the τ^{th} percentile frontier packet (assuming that frontier packets are ordered by time) in the chosen direction. This parameter allows us to tune the defense to produce super-traces that optimize some combination of bandwidth and latency overheads. The lower the value of τ, the lower the latency overhead (at the cost of bandwidth), and vice-versa. The range of τ is 1 to 100.

- The function `Find-Direction` returns +1 if more than $\frac{1}{6}$ of the frontier packets are upstream packets, else it returns -1. Function `Find-Time` returns the τ^{th} percentile time of frontier packets in direction P_D. `Find-Size` returns the maximum packet size in the frontier with time $\leq P_T$ and direction $= P_D$. Finally, function `Update-Frontiers` updates the frontier of each trace (F_i) to the last packet not covered by the current ST.

The algorithm is simple. The super-trace is computed over a set of input traces. In each iteration we add a trace from the least covered site into this input, until all sites have

Algorithm 6 Algorithm to compute the super-trace of a cluster

```
function INPUT-GEN(μ_min, τ, covmin, T, {R_1, ..., R_n})
    if coverage_covmin < μ_min then
        T ← T ∪ t, where t ∈ R_covmin;
        ST ← COMPUTE-ST(τ, T)
    else
        return ST
    end if
end function
function COMPUTE-ST(τ, T)
    ST ← ∅, F ← {1, ..., 1}
    while F ≠ {len_1 + 1, ..., len_m + 1} do
        P_D ←FIND-DIRECTION(T)
        P_T ←FIND-TIME(τ, T, P_D)
        P_S ←FIND-SIZE(T, P_T, P_D)
        ST ← ST||(P_D, P_T, P_S)
        F ← UPDATE-FRONTIERS(ST, T)
    end while
end function
```

satisfied the minimum coverage parameter μ_{min}.

Now, for each of these input traces, a counter (starting at the first packet) indicating the current frontier is maintained. We count the number of frontier packets in each direction. If more than $\frac{1}{6}$ of the packets are up-stream packets, we add an up-stream packet to the super-trace, otherwise, we add a down-stream packet. We use the parameter $\frac{1}{6}$ because in our input traces we found that the average ratio of up-stream to down-stream packets was $\frac{1}{6}$. The time at which this newly added packet is to be sent is set according to τ as described above. The size of the newly added packet is taken to be the maximum size (rounded to the nearest 50 bytes) of all the frontier packets in the chosen direction. The above process is repeated until the frontier of all the input traces have passed their final packet. The final trace ST is guaranteed to cover at-least μ_{min} of the traces of each site (although, in practice the average coverage turns out to be far higher).

6.2 Simulation Results

For our simulations, we collected 50 traces each from the Alexa top 500 functioning, non-redirecting web pages. The browser cache was cleared between page-loads. Next, each of the 50 traces for each site were then ranked by their *representativeness*, and clustering was done. For our simulations, we varied the number of clusters (k) in the range of 16 and 250, giving

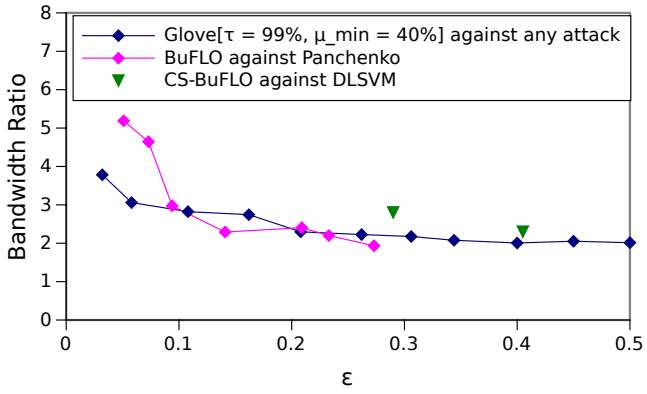
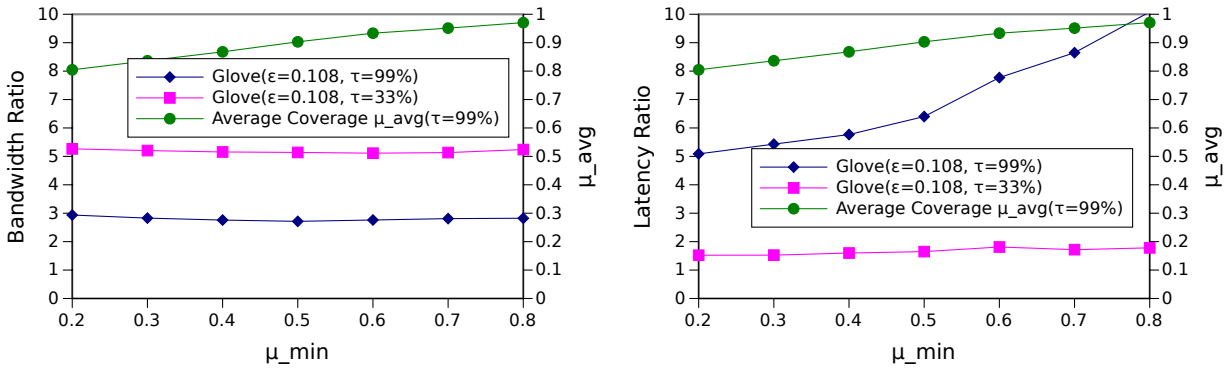


Figure 6.1: Bandwidth ratio as a function of the security parameter, ϵ, of various defense systems.

us defenses which were between 0.032 and 0.5 non-uniformly secure. Once the clusters were found, sets of super-traces were generated for each of the clusters while varying μ_{min} and τ. Finally, statistics corresponding to expected site coverage, bandwidth, and latency ratios were computed.

Overhead/Security Trade-off: Figure 6.1 compares the trade-off between overhead of Glove, BuFLO, and CS-BuFLO and the levels of security provided. The BuFLO results are taken from Dyer, et al., and therefore use $n = 128$. CS-BuFLO is evaluated using $n = 120$ and Glove is evaluated using $n = 500$ web pages. Note that in this plot, Glove is the only defense that provides information theoretic security *against any attacker*, while the data for BuFLO (obtained from [20]) and CS-BuFLO are relevant only against Panchenko and DLSVM attackers, respectively. Therefore, while the BuFLO and CS-BuFLO trade-offs might vary depending on the attacker, the Glove trade-off holds against any attacker. From the figure, it is clear that Glove provides significantly better trade-off costs than CS-BuFLO (vs. DLSVM, for any security level) and BuFLO (vs. Panchenko, when attack accuracy $\leq$ 10%). When security is less critical (attack accuracy $>$ 10%), however, Glove and BuFLO (vs. Panchenko) provide similar trade-offs.

Tunability of Glove: In Figure 6.2, we demonstrate the effect of varying the μ_{min} and τ parameters of Glove (on its bandwidth and latency ratios). In particular, the figures show that varying μ_{min} to increase resistance to dynamic content in web pages has little effect on the bandwidth overhead of the defense, while significantly increasing its latency overhead (when $\tau = 99\%$ – i.e., the defense is optimized for minimizing bandwidth costs). Further, we see that τ allows user experience tuning – i.e., reducing τ causes a significant drop in latency overheads (increasing the usability of the defense) at the cost of increased bandwidth.



(a) Bandwidth ratios

(b) Latency ratios

Figure 6.2: Overhead ratios and average coverage of Glove while varying μ_{min} and τ with $\epsilon = 0.108$.

Chapter 7

Discussion and Conclusion

7.1 DLSVM Attack

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, HTTPOS, and randomized pipelining, impose high costs but do not stop our attack.

We built an attack with 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 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, it outperforms 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 in Section 3.1.

Our results on the DLSVM attack support several conclusions:

Previously proposed 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 result, 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 + randomized pipelining data, our attack achieved between a 50% and 80% success rate. Furthermore, Figure 3.2 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. HTTPOS 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.

7.2 Theoretical Analysis

Our theoretical model clarifies the limits of website fingerprinting defenses. It establishes efficiency bounds that no defense can cross, giving an absolute benchmark for evaluating the efficiency of defenses. The lower bounds are surprisingly low, suggesting that it may be possible to build very efficient defenses. We also show that, in some contexts, randomized defenses offer no security or overhead advantage compared to deterministic defenses. This theoretical foundation also provides a framework for comparing schemes that offer different overhead/security trade-offs. Further, it allows conclusions to be drawn about open-world performance of attacks and defenses, based on their closed-world results. This greatly simplifies the experimental setup required to estimate open-world performance of attacks and defenses.

7.3 Congestion-Sensitive BuFLO

Congestion-Sensitive BuFLO is a high-security, moderate-overhead solution to website fingerprinting attacks. Compared to SSH and Tor, it achieves a better overhead/security trade-off, i.e. it uses its bandwidth efficiently to provide extra security. Our experiments also show that it has acceptable latencies. The padding schemes developed in CS-BuFLO, along with browser-coordination and the early-termination algorithm, can improve security with less overhead than previous stream padding schemes. Interestingly, we also found that padding from one end of a connection can sometimes be an efficient way to hide information about the data sent from the other side of the connection.

Since early termination does not seem to affect security, the padding results suggest that the padding performed while transmitting a website sufficiently hides the size of the website,

so that additional stream padding at the end of the transmission has little security benefit. Additional client padding does improve security, though – probably by obscuring the size of the final object requested by the client.

The lower bounds derived in Chapter 4.1.2 proved useful for comparing schemes. For example, without the lower bounds, it is difficult to determine whether Tor, SSH, or CS-BuFLO has the greatest efficiency in Figure 5.5(a), but it becomes obvious in Figure 5.5(c).

Overall, CS-BuFLO has better security than any previous defenses in our experiments, albeit at greater expense. It has the best overhead/security trade-off, as well.

CS-BuFLO's overhead/security trade-off is in the same range as the estimates Dyer obtained for BuFLO in their simulations. For example, Dyer, et al., reported that, in one configuration of BuFLO, bandwidth overhead was 200% and the Panchenko SVM had an 24.1% success rate on 128 websites. We found that CS-BuFLO with CTSP padding had an overhead of 180% on 120 websites, and that the Panchenko SVM had a success rate of 23.4%.

CS-BuFLO's congestion-sensitivity likely had little impact in these experiments, which were carried out on a fast local network, so that congestion was rare. However, CS-BuFLO's congestion-sensitivity means that, in a real deployment, it would have even better bandwidth overhead.

CS-BuFLO's latency overhead is approximately 3 in all our experiments. This is better than Tor's latency, although Tor has the additional overhead of onion routing, so no fair comparison is possible. We cannot compare with the latency estimates reported by Dyer, et al., because they gave only absolute latency values.

7.4 Glove

We presented Glove— a website fingerprinting defense that illustrates a promising new approach towards building efficient fingerprinting defenses. In particular, Glove is the first defense to demonstrate:

Information-Theoretic Security: Unlike previous defenses, Glove provides information-theoretic security guarantees. This is achieved by computing a single super-trace for each computed cluster and playing this each time a page contained in the cluster is loaded. As a result, in the absence of prior (or, outside) knowledge, any (current or future) website fingerprinting attackers success rate is bounded by the size of the computed cluster.

Good Overhead/Security Trade-off: The results illustrated in Figure 6.1 demonstrate the validity of our conjecture that using prior information about the structure of a web page to add cover traffic conservatively yields *better* website fingerprinting defenses. This

approach, used by Glove, results in it being more secure and efficient than any previously proposed SSH based defense.

High Tunability: Unlike previous defenses, Glove allows users to tune their browsing experience by adjusting the τ parameter. Lower τ values significantly reduce latency at the cost of bandwidth, and vice-versa. This is especially useful in current scenarios where it is likely the case that bandwidth costs are inconsequential, while even moderately increased user experienced latency may result in significantly decreased usability of the defense.

However, Glove also has limitations that may hamper its adoptability in the real world. We discuss these below.

Infrastructure Requirements: Glove greatly reduces its overhead by utilizing prior knowledge of web page structures. This requires the *Glove infrastructure* to be able to collect traces of defended web pages, cluster these pages, and compute corresponding super-traces. These tasks are fairly computationally intensive, making them infeasible for standard server side proxy nodes to perform on a regular basis. Instead, using a powerful dedicated central node to compute clusters/super-traces and distribute them to Glove proxies (e.g., via Tor bridges — since the security of Glove is independent of the secrecy of the computed super-traces) is more efficient. However, such a node may not be easily available in the real world. In the absence of such nodes, Glove has to fall back on less powerful server side proxies which take as input a list of URLs (from the client) and returns a single super-trace (to be played when the client loads a page in the input URLs). However, in these cases, while Glove retains its information-theoretic security guarantees, it is no longer able to use prior knowledge of web page structures to provide low overheads. To circumvent these problems, one may consider developing distributed clustering and super-trace algorithms for use with Glove.

Effect of Dynamic Content: Two types of dynamicity affect the Glove defense. First, the information-theoretic guarantees provided by Glove hold only when $\mu_{min} = 100\%$. This, however, is possible to guarantee only when web pages do not contain dynamic content (e.g., JS, AJAX, etc.). Second, when the structure of a Glove defended web page changes significantly, its trace may change to a large enough degree that less than $\mu_{min}\%$ of its page loads are covered by its current super-trace. In this case, re-clustering/super-tracing of all defended pages may be in order. While our observation is that this is rare, its occurrence does result in the need for the Glove infrastructure to occasionally perform this re-clustering, super-tracing, and distribution.

Bibliography

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

[2] Internet censorship. `http://en.wikipedia.org/wiki/Internet_censorship/`.

[3] Amazon web services - alexa top sites. `https://aws.amazon.com/alexatopsites/`, October 2013.

[4] Timothy G Abbott, Katherine J Lai, Michael R Lieberman, and Eric C Price. Browser-based attacks on tor. In *Privacy Enhancing Technologies*, pages 184–199. Springer, 2007.

[5] M.R. Albrecht, K.G. Paterson, and G. Watson. Plaintext recovery attacks against ssh. In *Security and Privacy, 2009 30th IEEE Symposium on*, pages 16–26, May 2009.

[6] Adam Back, Ulf Mller, and Anton Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In IraS. Moskowitz, editor, *Information Hiding*, volume 2137 of *Lecture Notes in Computer Science*, pages 245–257. Springer Berlin Heidelberg, 2001.

[7] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-resource routing attacks against tor. In *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society*, WPES '07, pages 11–20, New York, NY, USA, 2007. ACM.

[8] Aurelien Bellet, Amaury Habrard, and Marc Sebban. Good edit similarity learning by loss minimization. *Machine Learning*, 2012.

[9] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, 1994.

[10] George Bissias, Marc Liberatore, David Jensen, and Brian Levine. Privacy vulnerabilities in encrypted http streams. In *PETS*, 2006.

[11] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of service or denial of security? In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 92–102, New York, NY, USA, 2007. ACM.

[12] Philippe Boucher, Adam Shostack, and Ian Goldberg. Freedom systems 2.0 architecture. *White paper, Zero Knowledge Systems, Inc*, 2000.

[13] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981.

[14] Wei Dai. Pipenet 1.1. usenet post, august 1996, 1995.

[15] George Danezis. Traffic analysis of the HTTP protocol over TLS. `http://research.microsoft.com/en-us/um/people/gdane/papers/TLSanon.pdf`.

[16] George Danezis. The traffic analysis of continuous-time mixes. In David Martin and Andrei Serjantov, editors, *Privacy Enhancing Technologies*, volume 3424 of *Lecture Notes in Computer Science*, pages 35–50. Springer Berlin Heidelberg, 2005.

[17] The Internet Movie Database. `http://www.imdb.com/`.

[18] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[19] John R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 251–260, London, UK, UK, 2002. Springer-Verlag.

[20] 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 *IEEE Security and Privacy*, 2012.

[21] Matthew Edman and Paul Syverson. As-awareness in tor path selection. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 380–389. ACM, 2009.

[22] Nathan S. Evans, Roger Dingledine, and Christian Grothoff. A practical congestion attack on tor using long paths. In *Proceedings of the 18th Conference on USENIX Security Symposium*, SSYM'09, pages 33–50, Berkeley, CA, USA, 2009. USENIX Association.

[23] Facebook. `http://www.facebook.com/`.

[24] X. Fu, B. Graham, R. Bettati, and W. Zhao. On countermeasures to traffic analysis attacks. In *Information Assurance Workshop*, 2003.

[25] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[26] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Commun. ACM*, 42(2):39–41, February 1999.

[27] Xun Gong, Negar Kiyavash, and Nikita Borisov. Fingerprinting websites using remote traffic analysis. In *ACM CCS*, 2010.

[28] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naive-bayes classifier. In *ACM Workshop on Cloud Computing Security*, 2009.

[29] Andrew Hintz. Fingerprinting websites using traffic analysis. In *PETS*, 2002.

[30] Alan J. Hoffman and Joseph B. Kruskal. Integral boundary points of convex polyhedra. In Michael Jnger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 49–76. Springer Berlin Heidelberg, 2010.

[31] Nicholas Hopper, Eugene Y Vasserman, and Eric Chan-Tin. How much anonymity does network latency leak? *ACM Transactions on Information and System Security (TISSEC)*, 13(2):13, 2010.

[32] The Internet Society. *Hypertext Transfer Protocol – HTTP/1.1*, 1999.

[33] Rolf Jagerman, Wendo Sabée, Laurens Versluis, Martijn de Vos, and Johan A. Pouwelse. The fifteen year struggle of decentralizing privacy-enhancing technology. *CoRR*, abs/1404.4818, 2014.

[34] Jondonym: the anonymisation service. `http://anonymous-proxy-servers.net/`.

[35] S. Kadloor, Xun Gong, N. Kiyavash, T. Tezcan, and N. Borisov. Low-cost side channel remote traffic analysis attack in packet networks. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1–5, May 2010.

[36] Hemanta Kumar Kalita and Avijit Kar. Wireless sensor network security analysis.

[37] Leonard Kaufman and Peter Rousseeuw. Clustering by means of medoids. *Statistical Data Analysis Based on the L1-Norm and Related Methods*, 1987.

[38] I Keller and C Tompkins. An Extension of a Theorem of Dantzig's. *Linear Inequalities and Related Systems, Annals of Mathematics Studies*, 38:247–254, 1956.

[39] BrianN. Levine, MichaelK. Reiter, Chenxi Wang, and Matthew Wright. Timing attacks in low-latency mix systems. In Ari Juels, editor, *Financial Cryptography*, volume 3110 of *Lecture Notes in Computer Science*, pages 251–265. Springer Berlin Heidelberg, 2004.

[40] Marc Liberatore and Brian Neil Levine. Inferring the source of encrypted http connections. In *ACM CCS*, 2006.

[41] Zhen Ling, Junzhou Luo, Wei Yu, Xinwen Fu, Dong Xuan, and Weijia Jia. A new cell counter based attack against tor. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 578–589. ACM, 2009.

[42] Liming Lu, Ee-Chien Chang, and Mun Chan. Website fingerprinting and identification using ordered feature sequences. In *ESORICS*, 2010.

[43] 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.

[44] Nick Mathewson and Roger Dingledine. Practical traffic analysis: Extending and resisting statistical disclosure. In David Martin and Andrei Serjantov, editors, *Privacy Enhancing Technologies*, volume 3424 of *Lecture Notes in Computer Science*, pages 17–34. Springer Berlin Heidelberg, 2005.

[45] Jon McLachlan and Nicholas Hopper. On the risks of serving whenever you surf: vulnerabilities in tor's blocking resistance design. In *Proceedings of the 8th ACM workshop on Privacy in the electronic society*, pages 31–40. ACM, 2009.

[46] S.J. Murdoch and G. Danezis. Low-cost traffic analysis of tor. In *Security and Privacy, 2005 IEEE Symposium on*, pages 183 – 195, may 2005.

[47] StevenJ. Murdoch and Piotr Zieliski. Sampled traffic analysis by internet-exchange-level adversaries. In Nikita Borisov and Philippe Golle, editors, *Privacy Enhancing Technologies*, volume 4776 of *Lecture Notes in Computer Science*, pages 167–183. Springer Berlin Heidelberg, 2007.

[48] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33:31–88, March 2001.

[49] Gavin OGorman and Stephen Blott. Large scale simulation of tor. In *Advances in Computer Science–ASIAN 2007. Computer and Network Security*, pages 48–54. Springer, 2007.

[50] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *WPES*, 2011.

[51] Vasilis Pappas, Elias Athanasopoulos, Sotiris Ioannidis, and Evangelos P. Markatos. Compromising anonymity using packet spinning. In *Proceedings of the 11th International Conference on Information Security*, ISC '08, pages 161–174, Berlin, Heidelberg, 2008. Springer-Verlag.

[52] Adrian Perrig, John Stankovic, and David Wagner. Security in wireless sensor networks. *Commun. ACM*, 47(6):53–57, June 2004.

[53] Mike Perry. Experimental defense for website traffic fingerprinting. https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting, September 2011.

[54] Mike Perry. A critique of website traffic fingerprinting attacks. `https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks`, November 2013.

[55] Andreas Pfitzmann and Marit Hansen. Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management-a consolidated proposal for terminology. *Version v0*, 31:15, 2008.

[56] R. Pries, Wei Yu, Xinwen Fu, and Wei Zhao. A new replay attack against anonymous communication networks. In *Communications, 2008. ICC '08. IEEE International Conference on*, pages 1578–1582, May 2008.

[57] Jean-Franois Raymond. Traffic analysis: Protocols, attacks, design issues, and open problems. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies*, volume 2009 of *Lecture Notes in Computer Science*, pages 10–29. Springer Berlin Heidelberg, 2001.

[58] Juha Salo. Recent attacks on tor. *Aalto University*, 2010.

[59] Andrei Serjantov and Peter Sewell. Passive attack analysis for connection-based anonymity systems. In Einar Snekkenes and Dieter Gollmann, editors, *Computer Security ESORICS 2003*, volume 2808 of *Lecture Notes in Computer Science*, pages 116–131. Springer Berlin Heidelberg, 2003.

[60] Paul Seymour. Decomposition of regular matroids. *Journal of Combinatorial Theory, Series B*, 28:305–359, 1980.

[61] 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.

[62] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on ssh. In *Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10*, SSYM'01, Berkeley, CA, USA, 2001. USENIX Association.

[63] Qixiang Sun, Daniel R. Simon, Yi-Min Wang, Wilf Russell, Venkata N. Padmanabhan, and Lili Qiu. Statistical identification of encrypted web browsing traffic. In *IEEE Security and Privacy*, 2002.

[64] Tor project: Anonymity online. `https://www.torproject.org/`, August 2011.

[65] Carmela Troncoso and George Danezis. The bayesian traffic analysis of mix networks. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 369–379, New York, NY, USA, 2009. ACM.

[66] Lisa Vaas. Fbi used drive-by downloads to track child porn suspects hidden on tor. `http://nakedsecurity.sophos.com/2014/08/06/fbi-used-drive-by-downloads-to-track-child-porn-suspects-hidden-on-tor/`, August 2014.

[67] Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., 1995.

[68] AJ. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269, April 1967.

[69] 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.

[70] D. Welch and S. Lathrop. Wireless security threat taxonomy. In *Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society*, pages 76–83, June 2003.

[71] Andrew M. White, Austin R. Matthews, Kevin Z. Snow, and Fabian Monrose. Phonotactic reconstruction of encrypted VoIP conversations: Hookt on fon-iks. In *Proceedings of the 32nd IEEE Symposium on Security and Privacy*, 2011.

[72] Charles Wright, Fabian Monrose, 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.

[73] Charles V. Wright, Scott E. Coull, and Fabian Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *NDSS*, 2009.

[74] Matthew K Wright, Micah Adler, Brian Neil Levine, and Clay Shields. Passive-logging attacks against anonymous communications systems. *ACM Transactions on Information and System Security (TISSEC)*, 11(2):3, 2008.

[75] Shui Yu, Wanlei Zhou, Weijia Jia, and Jiankun Hu. Attacking anonymous web browsing at local area networks through browsing dynamics. *The Computer Journal*, 2011.

[76] Fan Zhang, Wenbo He, Xue Liu, and Patrick G. Bridges. Inferring users' online activities through traffic analysis. In *ACM Conference on Wireless Network Security*, 2011.

[77] Yang Zhang. Effective attacks in the tor authentication protocol. In *Network and System Security, 2009. NSS'09. Third International Conference on*, pages 81–86. IEEE, 2009.

[78] Yongguang Zhang, Wenke Lee, and Yi-An Huang. Intrusion detection techniques for mobile wireless networks. *Wirel. Netw.*, 9(5):545–556, September 2003.

Appendices

A. HMMs for Facebook and IMDB

The Facebook HMM has 7 states:

1. homepage_cold
2. profile_cold
3. homepage_scroll
4. profile_scroll
5. login_page
6. homepage_warm
7. profile_warm

with transition probabilities

$$\begin{bmatrix} 0 & 0.6 & 0.2 & 0 & 0.1 & 0.1 & 0 \\ 0 & 0 & 0 & 0.2 & 0.1 & 0.4 & 0.3 \\ 0 & 0.6 & 0.2 & 0 & 0.1 & 0.1 & 0 \\ 0 & 0 & 0 & 0.2 & 0.1 & 0.4 & 0.3 \\ 0.9 & 0 & 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0.2 & 0.2 & 0 & 0.1 & 0.1 & 0.4 \\ 0 & 0 & 0 & 0.2 & 0.1 & 0.4 & 0.3 \end{bmatrix}$$

The initial probability of all states is 0.142, except for the login state, which is 0.148.

The IMDB HMM has 8 states:

1. homepage_cold

2. searchpage_cold
3. moviepage_cold
4. starpage_cold
5. homepage_warm
6. searchpage_warm
7. moviepage_warm
8. starpage_warm

with transition probabilities

$$\begin{bmatrix} 0 & 0.6 & 0.2 & 0.2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.4 & 0.4 & 0.1 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0.2 & 0.1 & 0.1 & 0.2 & 0.3 \\ 0 & 0 & 0.2 & 0.1 & 0.1 & 0.1 & 0.3 & 0.2 \\ 0 & 0.1 & 0.1 & 0.1 & 0.1 & 0.2 & 0.2 & 0.2 \\ 0 & 0 & 0 & 0 & 0.1 & 0.1 & 0.4 & 0.4 \\ 0 & 0 & 0 & 0 & 0.1 & 0.1 & 0.3 & 0.5 \\ 0 & 0 & 0 & 0 & 0.1 & 0.1 & 0.5 & 0.3 \end{bmatrix}$$

All states have initial probability 1/8.

B. Lower Bound Proofs

Suppose websites $w_1, \ldots, w_n$ have sizes $s_1 < s_2 < \ldots < s_n$. Let $S = \{s_1, \ldots, s_n\}$. For any defense, D, let p_{ij} be the probability that D transmits j bytes during a load of website w_i.

Since, in a closed-world experiment, each website occurs with probability $1/n$, the bandwidth cost of D is

$$\sum_{j=1}^{\infty} \sum_{i=1}^{n} \frac{1}{n} j p_{ij}$$

and the non-uniform success probability of A_S is

$$\sum_{j=1}^{\infty} \frac{\max_i p_{ij}}{\sum_i p_{ij}} \cdot \frac{\sum_i p_{ij}}{n} = \sum_{j=1}^{\infty} \frac{\max_i p_{ij}}{n}$$

We derive lower bounds on the bandwidth cost of D by computing the matrix of p_{ij} values that minimize the above bandwidth cost function while still satisfying the above security constraint. Recall that, since D is assumed not to compress or truncate web pages, $p_{ij} = 0$ for $j < s_i$.

The overall structure of the proof for non-uniform security is

- Constrain the structure of the optimal p_{ij} so that we can formulate the optimization problem as a linear program. (see Lemma 1).
- Prove that the linear program has an integral solution, so that the optimal solution is equivalent to a function $f : S \to S$ satisfying certain constraints (see Lemma 2).
- Prove that f is monotonically increasing (see Lemma 3).

The lower bound for uniform security is similar. We first prove that there exists a similar function f for any deterministic uniformly secure defense, and then apply Lemma 3.

Lemma 1. *Let p_{ij} be the probabilities that minimize the bandwidth cost while meeting the security requirement.*

1. *$p_{ij} = 0$ unless $j \in \{s_1, \ldots, s_n\}$.*
2. *If $p_{ij} < \max_k p_{kj}$, then for all $j' > j$, $p_{ij'} = 0$.*
3. *For all j, $p_{kj} \leq p_{k+1,j}$ for $k \in [1, i]$, where $s_i \leq j < s_{i+1}$.*

Proof.

1. Suppose $p_{\ell j} \neq 0$ where $s_k < j < s_{k+1}$. Then we can make a more efficient and no less secure by replacing p_{is_k} with $p_{is_k} + p_{ij}$ for all i and setting $p_{ij} = 0$ for all i. This will have lower bandwidth cost because $s_k < j$. This will not violate the constraint that, for all i and $j' < s_i$, $p_{ij'} = 0$, because, if $p_{ij} \neq 0$ before the change, then $s_i \leq j$, so $s_i \leq s_k$. This will not worsen security because $\max_i(p_{is_k} + p_{ij}) \leq \max_i p_{is_k} + \max_i p_{ij}$.

2. Suppose otherwise. Let $t = \min(\max_k p_{kj} - p_{ij}, p_{ij'})$. Note $t \neq 0$. Thus we can construct a more efficient and no less secure defense by replacing p_{ij} with $p_{ij} + t$ and $p_{ij'}$ with $p_{ij'} - t$.

3. Suppose $p_{kj} > p_{k+1,j}$ for some $k \in [1, i]$, where $s_i \leq j < s_{i+1}$. This implies that $p_{k+1,j} < \max_i p_{ij}$. By Item 2, $p_{k+1,j'} = 0$ for all $j' > j$.

   Thus we must have that: $\sum_{j'=s_{k+1}}^{j} p_{k+1,j'} = 1$.

   This also implies that $p_{kj} \neq 0$. Thus, by Item 2, $p_{kj'} = \max_i p_{ij'}$ for all $j' \in \{s_k, \ldots, j-1\}$. This implies that $\sum_{j'=s_k}^{j} p_{kj'} > \sum_{j'=s_{k+1}}^{j} p_{k+1,j'} = 1$, a contradicition.

Since p_{ij} is non-zero only if $j \in \{s_1, \ldots, s_n\}$, we can relabel the p_{ij} to be the probability that the defense transmits s_j bytes during a load of website w_i.

Lemma 1, Item 3 implies that $\max_i p_{ij} = p_{ii}$, so the security constraint can be re-written as $\sum_{i=1}^{n} p_{ii} \leq \epsilon n$.

Now that the security constraint is a linear function of the p_{ij} variables, we can formulate a linear program for computing the optimal p_{ij} values:

$$\text{minimize} \sum_{i=1}^{n} \sum_{j=i}^{n} p_{ij} s_j \qquad \text{(the bandwidth cost)}$$

subject to the constraints

$$\begin{array}{ll} (a) & \sum_{i=1}^{n} p_{ii} \leq \epsilon n \qquad (\epsilon \text{ non-uniform security}) \\ (b) & \sum_{j=i}^{n} p_{ij} = 1 \qquad (p_{ij} \text{ are probabilities}) \\ (c) & 0 \leq p_{ij} \leq 1 \end{array}$$

Lemma 2. *The above linear program has an integral solution.*

Proof. Linear programs with Totally Unimodular (TU) constraint matrices and integral objective functions have integral solutions [30]. We prove that the constraint matrix, A (derived by the constraints (a), (b), and (c) of the above LP), is TU. To prove TU-ness of A, it is sufficient to prove the following [38]: (i) Every column contains at-most 2 non-zero entries, (ii) Every entry is 0, 1, or -1, (iii) If two non-zero entries in any column of A have the same sign, then the row of each belongs in two disjoint partitions of A.

Since the set of TU matrices is closed under the operation of adding a row or column with at-most one non-zero entry [60], we may delete the $2n$ rows of A corresponding to constraint (c) and prove that the remaining constraint matrix A' satisfies the TU conditions (i) - (iii).

Observe the following properties of A':

- There are n rows (WLOG, rows 1 to n) induced by the constraint (a). These are such that: $A_{i,(i-1)n}, \ldots, A_{i,in-1} = 1$, $\forall i \in \{1, \ldots, n\}$ and 0 for all other entries. Therefore, each column of the partition B composed of these n rows contains only a single non-zero entry (i.e., +1).

- There is only 1 row (WLOG, row $n+1$) induced by the constraint (b). This row has the form: $A_{n+1,j} = 1$, $\forall j \in \{1^2, \ldots, n^2\}$ and 0 for all other entries. Each column of the partition C composed of this single vector may contain only a single non-zero entry (i.e., +1).

From the above properties, it is clear that matrix A' is TU since: Each column contains at-most 2 non-zero entries (+1) and it may be partitioned into matrices B and C such that condition (iii) is satisfied. Therefore, the matrices A' and A are TU and the LP describing A has only integral optima.

In an integral solution of the linear program, all the probabilities are 0 or 1, so the solution is equivalent to a function $f : S \to S$ satisfying

- $|f(S)| \leq \epsilon n$.
- $\sum_{i=1}^{n} f(s_i) / \sum_{i=1}^{n} s_i \leq \text{BWRatio}_D(W)$.

We now show there is a similar function for any deterministic uniformly secure defense D. Set $f(s_i) = b_i$ where b_i is the number of bytes transmitted when the defense D loads website w_i. Since D does not compress or truncate websites, we must have $b_i \geq \max_{s \in f^{-1}(b_i)} s$ for all i. Observe that we can assume $b_i = \max_{s \in f^{-1}(b_i)} s$ without harming security or efficiency, so that $f : S \to S$. Thus f satisfies the security constraint $\min_i |f^{-1}(s_i)| \geq 1/\epsilon$, and $\sum_{i=1}^{n} f(s_i) / \sum_{i=1}^{n} s_i \leq \text{BWRatio}_D(W)$.

Lemma 3. *The mapping function f corresponding to an optimal non-uniformly ϵ-secure defense, or a deterministic uniformly ϵ-secure defense, is monotonic.*

Proof. Consider any partition of $\{s_1, \ldots, s_n\}$ into sets $S_1, \ldots, S_k$. Let $m_i = \max_{s \in S_i} S_i$. Without loss of generality, assume $m_1 \leq m_2 \leq \cdots \leq m_k$. Now consider the monotonic allocation of traces into sets $S_1^*, \ldots, S_k^*$ where $|S_i^*| = |S_i|$. Let $m_i^* = \max_{s \in S_i^*} s$. Observe that $m_i^* \leq m_i$ for all i, i.e. the new allocation has lower bandwidth.

Since the number of sets in the partition and the sizes of those sets are unchanged, this new allocation has the same uniform and non-uniform security as the original, but lower bandwidth. Hence the optimal f must be monotonic.