Link Prefetching: A Defense Against Website Fingerprinting on Tor *

Vaibhav Sharma
sharm361@umn.edu

Taejoon Byun
taejoon@umn.edu

Elaheh Ghassabani
ghass013@umn.edu

Se Eun Oh
seoh@umn.edu

Department of Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55454

ABSTRACT

This paper is written to get an A in the CSCI5271 course. PERIOD.

This content will be edited later. We plan to explore the area of website fingerprinting in anonymization networks starting with the paper Website Fingerprinting in Onion Routing Based Anonymization Networks.

Keywords

Website fingerprinting, anonymity, encrypted traffic, Tor

1. INTRODUCTION

Anonymizing networks are privacy technologies that provide a mancinism to anonymize internet communications so as to protect users from network eavesdroppers. Although such systems are able to hide the communication (including both routing information and content), an attacker is still able to obtain different information by analyzing the network traffic. Network analysis can provides very rich information about message length, timing, and frequency by which an attacker can easily identify the communicating parties, and therefore bypass bypass an anonymizing system. This problem is known as Website Fingerprinting (WF) attack, where the adversary attempts to recognize the encrypted traffic patterns of specific web pages without using any other information [10, 12].

This report is organized as follows;

2. WEBSITE FINGERPRINTING ON TOR

This section explains the background about Tor, an anonymity network, and the website fingerprinting attack which can neutralize the anonymity that Tor provides.

*This report is submitted as a partial fulfillment of *CSCI5271: Introduction to Security* course.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

2.1 Tor

The Onion Router (Tor) is one of the most popular anonymizing networks, which distributes user's communications over several places on the Internet. The distribution helps user's final destination not to be linked to a single point. In other words, packets take a random path through several relays, instead of taking a direct route. These layers cover the user path such that it will not be possible to tell where the data comes from, or where it is going. Such a pathway in Tor is called a private network pathway. The Tor client application is responsible of creating this private pathway by incrementally building a circuit of encrypted connections. The circuit is constructed hop by hop, which means each individual relay is only aware of the previous and next hop. That is to say, individual relays never can identify the complete path [2, 4]

If an attacker can see both ends of the private pathway, Tor will fail. For example, suppose the attacker watches the Tor relay to which the user enters the network, and also watches the website he visit; In this case, the attacker can correlate volume and timing information on the two sides. To cope with this problem, Tor came up with the idea of "entry guards", where each Tor client randomly selects a few relays as entry points for her first hop. If those relays are not observed by an attacker, the user is secure. But, needless to say, there is always a probability of losing anonymity [2].

2.2 Website Fingerprinting

Tor does not provide protection against all anonymity problems. In addition WF attack is still able to bypass its privacy mechanism. In the case of Tor, WF attack would take place between the user and the Guard node, or at the Guard node itself. In general, attack models can be divided into two categories: "closed world" and "open world" scenarios. This section describes these attack models.

In the closed world model, the classifier can successfully recognize only web pages that it has already been trained on. Therefore, in this case, we deal with a small set of censored web pages. The open world scenario is when the ability of the adversary can recognize censored pages that might have not seen before [1].

Figure 1 illustrates the attack model of website fingerprinting.

Figure 1: Website Fingerprinting Attack Model

3. RELATED WORK

There are a decent number of works on bypassing anonymizing systems, especially Tor, on which this section provides an overview.

3.1 Fingerprinting Attacks

The idea of extracting information about content of encrypted SSL packets dates back to 1996 [15]. Later in 2002, Hintz presented an attack on an encrypting web proxy calling it website fingerprinting. The attack successfully exploited a traffic analysis-based vulnerability to detect which website a particular user is surfing [9]. Since then, a lot of research has been conducted in this area trying either to propose a more realistic attack, or a more effective defense mechanism.

One interesting study has been done in [8], where authors applied WF attack on different anonymizing techniques, including OpenSSH, OpenVPN, Stunnel, Tor. They tried to attack 775 different websites. The result presents a low detection rate in the closed world attack model (only a 3% success rate for 775 pages). They used Multinomial Naive Bayes classifier focusing only on packet sizes. Later on, this attack has been improved with around 90% accuracy for 100 wbpages in [16, 3]. In [3], authors could defeat ad hoc defense mechanisms describing a new defense scheme that provides provable security properties. Penchenko et al. [14] were also among the first to report website fingerprinting attacks with reasonable accuracy on Tor. They provided a sufficient understanding of the feature set and classification framework required for this attack. In fact, they carried out a comprehensive study of WF on Tor, which considers both closed world and open world attack models.

Generally speaking, attacks on anonymizing networks took a variety of approaches; some of the attacks tries to discover the identity of the anonymous user, others focus on uncovering the private pathway, and others attempt to identify the servers users interact with [3].

3.2 Defense Mechanisms

Usually, defense mechanisms are developed at the IP/TCP level by changing the pattern of traffic. For example, they split packets into multiple packets, or insert fake packets into the traffic, or involve padding packets. A study performed in [7] shows that transmission at random intervals could be a defense against analysis-based attacks. Another defense technique proposed in [17] uses some tricks to change a traffic pattern in a way that it looks like another pattern. However, this method does not split or disordered packets. Therefore, attacks that work without packet size information can easily defeat this mechanism. In [11], Luo et al. proposed a collection of tricks at the HTTP/TCP level as a defense mechanism against some analysis attacks. For example, they changed window sizes and order of packets in the TCP stream. Also, at the HTTP level, they tried to insert some extra data into HTTP GET headers, generates some

3.3 Criticism

4. LINK PREFETCHING AS A DEFENSE

A classifier used in a website fingerprinting attack can distinguish the differnce between two classes of packets when the difference is consisntent between them [**?**]. Thus, intuitively, a classifier would work the best if the difference in the same class is minimal and the difference among heterogeneous classes is high at the same time. However, when variance among fingerprints in the same class is high, a classifier would not be able to characterize it clearly or at least the detection rate using the classifier would be low. This is the basic idea of using prefetching as a defense mechanism; our conjecture is that we can use prefetching to manipulate the number and size of incoming and outgoing packets in order to increase the variance among packets in the same class.

This section explains the concept of link prefetching, discusses its effect on website fingerprints and argue its possible usage as a defense mechanism.

4.1 Link Prefetching

Link prefetching is a HTML syntax that gives the web browser hints about which page the user is most likely to visit in a near future [5, 6]. The pages and resources to pre-fetch are specified in the web page so that the web browser can silently load them (or pre-fetch them) after an idle time. Since the pre-fetch happens only after the page is fully loaded, it does not sacrifice the loading time of the requested web page. Moreover, it can save the loading time for the pre-fetched pages and thus improve the user experience by caching the *future* contents. It was first suggested by Mozilla Foundation in 2003 and supported by most modern browsers nowadays.

The resources to prefetch can be simply specified in *HTML* using a `link` tag [13]. For example, a `link` tag `<link rel="prefetch" href="/page2.html">` tells the browser to prefetch a *html* file named *page2.html*. Resources other than a *HTML* web page can also be pre-fetched similarly using the same syntax. There are also some variations for different types of prefetching – DNS prefetching, which is specified as `<link rel="dns-prefetch" ..>`, is supported by *Mozilla Firefox* and *Google Chrome*, and `<link rel="prerender" ..>` also does the same job as `prefetch` in *Google Chrome* and *Microsoft Internet Explorer*.

Figure 2 illustrates how prefetching actually works in a browser (*Google Chrome*). This page is set up arbitrarily by the authors to demonstrate link prefetching, and contains a link pre-fetch tag that specifies a big image (the image on the left side labeled as *prefetch*). When the prefetching is off, this image shall be requested only when a user puts his mouse cursor on it (mouse over). However, it can be seen on the network timeline (right bottom) that the image is prefetched right after loading the page. This is indicated by a long blue bar on the second row for the file named "*Very-high...*", and it is long because the size of the file is relatively big (*3.5 MB*) that it took a longer time to download. Please also note that the time it took for loading the image when the user actually requested (by putting his mouse on it) was very short, because the image had already been pre-fetched

that the browser merely loaded it from the cache (as shown in the *size* column on the fourth row).

4.2 The Effect of Prefetching on Website Fingerprint

4.2.1 An Example of prefetching packet sequence

Figure 3 illustrates a website fingerprint in terms of inter-packet timing, where the *x-axis* represents time in seconds and the *y-axis* represents the number of packets captured during a certain time interval (*500 ms*). The solid line depicts the packets captured while prefetching was enabled, and the dashed line depicts when the pre-fetch was disabled in the browser settings. Both the cases are captured for the same web page, which is the first page of `wired.com`. However, the number of packets are slightly different for each case (4055 for the *off* case while 4218 for the *on* case), because the prefetch-on case obviously requests more resources for caching purpose. The elapsed time for loading the whole page was also different, but this variance is mainly due to the difference in network condition because all the packets are more scattered throughout time compared to the prefetch-off case. When we ignore the speed difference, we can see that the four peaks of the two lines are roughly the same that if we capture the packets multiple times for the same website, we can characterize how the *fingerprint* of a specific website looks like. Please note also that inter-packet timing is only one of many features for characterizing website fingerprint.

4.3 Fingerprint Features

Since all the traffic on Tor is encrypted, fingerprinting attacks blindly analyze a sequence of packets without knowing its contents, and extract *features* which characterizes a fingerprint. The attacker then trains his/her classifier on multiple packet sequences from the set of website, for a set of features that attacker determined to use. In this subsection, we summarize the definition of the four most popular features defined by Cai et. al. [**?**] to discuss the effect of prefetching on this features in the following subsection.

Packet Sequence: A packet sequence P can be written as $P = \langle (t_1, l_1), (t_2, l_2), ..., (t_n, l_n) \rangle$ [**?**], where t_i is the time relative to t_0 which is 0, and l_i is the byte length of the i-th packet.

Unique Packet Lengths: Most packets

$$(\exists L \in P_l | L \notin P_l') \vee (\exists L \in P_l' | L \notin P_l) \quad (1)$$

Packet Length Frequency: When $n_L(P_l)$ is the number of times packet length L appears in P_l,

$$\exists L | n_L(P_l) \neq n_L(P_l') \wedge n_L(P_l) > 0 \wedge n_L(P_l') > 0 \quad (2)$$

In other words, P and P' have different packet length frequencies iff there exists a packet with length L that appears different times in the two sequences.

Packet Ordering: When M_l is the multiset of packet lengths in P_l without ordering, two packet sequences P and P' have different packet ordering iff:

$$M_l = M_l' \wedge P_l \neq P_l' \quad (3)$$

Interpacket Timing:

$$\exists i, 1 \leq i \leq min(|P|, |P'|) : (P_t)_i \neq (P_t')_i \quad (4)$$

Table 1: Experiment design to answer *RQ1*

victim \attacker	prefetch on	prefetch off
prefetch on	(1)	(2)
prefetch off	(3)	(4)

4.3.1 The Effect of Prefetching on the Features

Existing fingerprinting attacks extract *features* from a sequence of packets to characterize a distinct characteristic of a website. The most popular features that have been used in the previous works for classification are unique packet lengths, packet length frequency, packet ordering, and interpacket timing. Existing works on website fingerprinting considers a sequence Packet lengths Packet length frequency Packet ordering. Interpacket timing

Packet sequence.

5. EXPERIMENTAL EVALUATION

Experimental result comes here.

5.1 Research Question

1. **RQ1**: Does prefetching itself provide an extra degree of defense?
2. **RQ2**: Can prefetching be used as a browser-side defense mechanism?

1. We speculate that prefetching itself might provide extra defense because of the extra packets. It can also be the case however, prefetching websites are more vulnerable to fingerprinting because of the extra prefetch packets that shows distinct prefix.
2. This case is unlikely since prefetching is on by default. We assume that victims will more likely be using Tor under the default setting.
3. This case is what we are most curious about, whether a victim can confuse an attacker by simply turning prefetching setting off of his browser.
4. This case simulates a situation where a victim is loading any other websites that does not prefetch any resource. This can be used as a comparison case.

5.2 Effects of Pre-Fetching on Fingerprinting

In this section, we will write about our experiments. We are planning to conduct two sets of experiments. If we consider the network traffic, the number of packages go upstream depends on the number of pre-fetching requests, and the number of downstream packages coming depends on the size of resources that should be pre-fetched. Therefore, it is obvious that pre-fetching would affect the fingerprint of the traffic of a particular website. *should be completed.*

5.3 Investigate Pre-Fetching Effects on top 60 Popular Websites

We are running experiments to see how pre-fetching affects the websites' fingerprints. After doing some search on top popular websites, we put together a small crawler by which we learnt that only around 60 of all 6000 websites

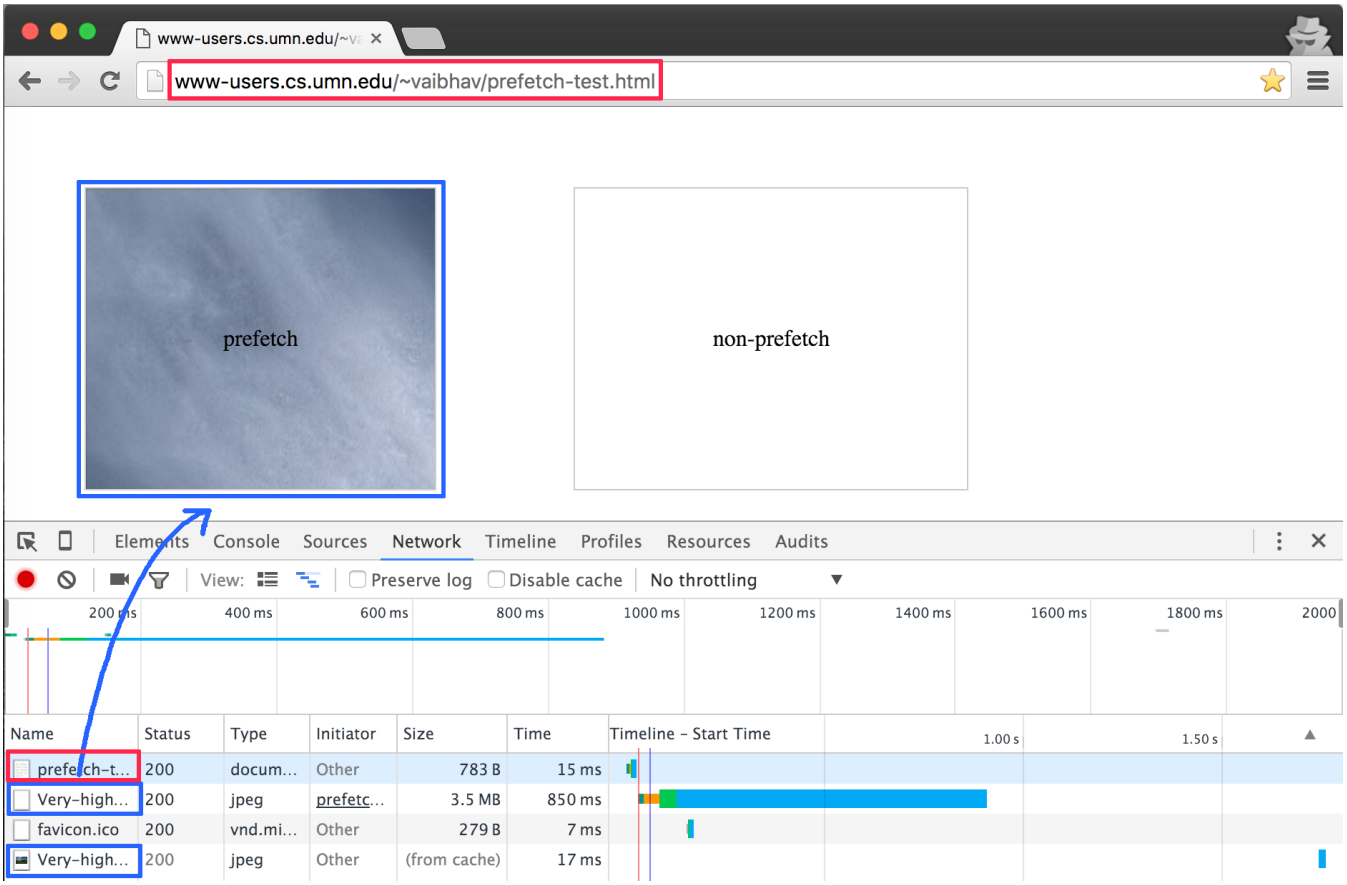

Figure 2: Network timeline showing pre-fetch

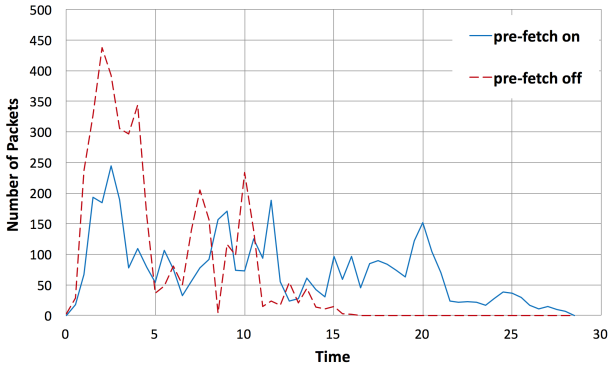

Figure 3: A website fingerprint for pre-fetch on/off cases

are use pre-fetching mechanism. We are capturing traffic of these websites in two different modes: 1) with enabled pre-fetching, and 2) with disabled pre-fetching. We are working on feature extraction, and about to decide which classifiers to use for the learning phase. Ultimately, we plan to conduct two sets of experiments. One sort of experiment is to compare two series of the captured packets and find the accuracy number with the help of a classifier, by which our goal is to provide an evidence to see if pre-fetching really affects fingerprints of websites. So, if the result will be positive, we will perform another set of experiment, which kind of simulates a sub set of those 60 websites. Then, we will see how (altering) the size of pre-fetching affects fingerprinting attacks/ defense mechanisms.

5.4 Effect of Pre-Fetching Packets Size on Fingerprinting Attacks

Here, we will explain our second experiment. We will simulate a sub set of webpages we investigated in the previous experiment. Then, we will equip them with a mechanism so that they can affect the downstream traffic and finally their fingerprint. Then, we will analyze the result to see how this idea contributes to the effectiveness of attacks and defense techniques.

6. DISCUSSION

We'll find what to discuss later on after finishing the experiment section.

7. CONCLUSIONS

What we have done in this work is so cool and awesome. What you may criticize will all be put here as "future work".

This section will conclude the result of our experiments. Finally we will provide some evidence to show how pre-fetching affect fingerprinting attacks. Based on our result, we are planing to suggest some defense mechanisms.

8. ACKNOWLEDGMENTS

The authors appreciate Professor Stephen McCamant for telling geeky jokes in classes all the time. This is a research project for CSCI5271, University of Minnesota.

9. REFERENCES

[1] A critique of website traffic fingerprinting attacks. https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks, 2015. Accessed: 2015-11-1.

[2] Tor project. https://www.torproject.org/, 2015. Accessed: 2015-12-12.

[3] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 605–616. ACM, 2012.

[4] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[5] D. Fisher. Link prefetching FAQ, 2003.

[6] D. Fisher and G. Saksena. Link prefetching in mozilla: A server-driven approach. In *Web content caching and distribution*, pages 283–291. Springer, 2004.

[7] X. Fu, B. Graham, R. Bettati, and W. Zhao. On countermeasures to traffic analysis attacks. In *Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society*, pages 188–195. IEEE, 2003.

[8] D. Herrmann, R. Wendolsky, and H. Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 31–42. ACM, 2009.

[9] A. Hintz. Fingerprinting websites using traffic analysis. In *Privacy Enhancing Technologies*, pages 171–178. Springer, 2003.

[10] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 263–274. ACM, 2014.

[11] X. Luo, P. Zhou, E. W. Chan, W. Lee, R. K. Chang, and R. Perdisci. Httpos: Sealing information leaks with browser-side obfuscation of encrypted flows. In *NDSS*, 2011.

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

[13] M. Nottingham. Rfc5988: Web linking. Technical report, 2010.

[14] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pages 103–114. ACM, 2011.

[15] D. Wagner, B. Schneier, et al. Analysis of the ssl 3.0 protocol. In *The Second USENIX Workshop on Electronic Commerce Proceedings*, pages 29–40, 1996.

[16] T. Wang and I. Goldberg. Improved website fingerprinting on tor. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 201–212. ACM, 2013.

[17] C. V. Wright, S. E. Coull, and F. Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *NDSS*, 2009.