Link Prefetching: A Defense Against Website Fingerprinting on Tor *

Vaibhav Sharma
vaibhav@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.A++ actually

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 provide very rich information about message length, timing, and frequency by which an attacker can easily identify the communicating parties, and therefore bypass an anonymizing system. This problem is known as Website Fingerprinting (WF) attack, where an adversary attempts to recognize the encrypted traffic patterns of specific web pages without using any other information [16, 18].

This report is organized as follows;

2. WEBSITE FINGERPRINTING ON TOR

This section provides a background on The Onion Router (Tor) and the website fingerprinting attack which can neutralize the anonymity that Tor provides.

2.1 Tor

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. Tor can be considered as an overlay network on top of the internet which directs TCP streams to different proxies. In this network, 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. In order to build circuits, Tor makes use of authenticated DH key exchange encrypted with symmetric keys [5, 10]. Figure 1 demonstrates how Tor works.

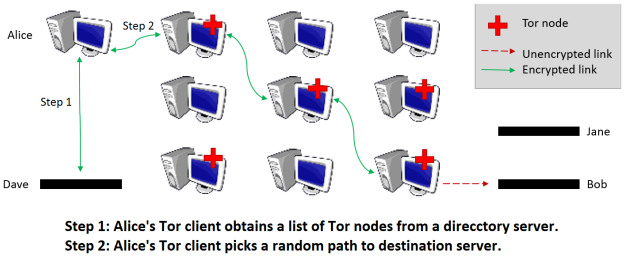

Figure 1: The Onion Router (Tor) [5]

The design described above aims to support a variety of anonymity goals. However, 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 she visits; in this case, the attacker can correlate volume and timing information on the two sides. To cope with this problem, Tor came

*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

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 [5].

2.2 Website Fingerprinting

Tor does not provide protection against all anonymity problems. In addition, the 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 adversary is capable of recognizing censored pages that might have not seen before [2].

Figure 2 illustrates the attack model of website fingerprinting. We also consider the WF adversary as local and passive, which means although the adversary is able to eavesdrop on user's traffic, he cannot modify it. It should be pointed out that the adversary will not be able to decrypt the traffic. Otherwise, he would not bother lunching a WF attack [16].

Figure 2: Website Fingerprinting Attack Model [16]

Juarez et al. in [16] divided the WF attack into two different types based on the number of the users it targets. In the first category, called *targeted*, the adversary aims to obtain the browsing activity of a specific victim. In this case, since the adversary is able to train the classifier using the information of the victim, the attack is more successful. The adversary may even have enough information and background knowledge about the user and her behavior. So, it could be possible to guess and emulate the user configuration. Therefore, the accuracy of the classifier would increase. The second type of the attack is called *non-targeted* or *dragnet surveillance*, where a set of users, instead of one, are targeted by the attacker. In this case, ISP, entry guard, or Internet exchanges serves the attacker. In other words, the attacker is able to intercept the traffic of many users by using such servers. A classifier in the non-targeted attack is trained based on a specific setting and used for all communications.

3. RELATED WORK

This section surveys attacks/ defenses on/against anonymizing systems, especially Tor.

3.1 Fingerprinting Attacks

The idea of extracting information about the content of encrypted SSL packets dates back to 1996 [23]. 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 [15]. 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 [14], where authors lunched 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 [24, 9]. In [9], authors could defeat ad hoc defense mechanisms describing a new defense scheme that provides provable security properties. Penchenko et al. [20] 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 take 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 [9].

3.2 Defense Mechanisms

Defense mechanisms are usually developed at the IP/TCP level by changing the pattern of traffic. For example, they split packets into multiple packets, insert fake packets into the traffic, or involve padding packets. A study performed in [13] shows that transmission at random intervals could be a defense against analysis-based attacks. Another defense technique proposed in [25], called traffic morphing, 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 [17], 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 the order of packets in the TCP stream. Besides, at the HTTP level, they tried to insert some extra data into HTTP GET headers, generate some irrelevant HTTP request, and re-order requests. The Tor community also introduced "randomized pipelining" [21] as a defense mechanism, where the browser loads the web content in a random order. Despite such techniques, the proposed attack in [9] managed to successfully recognize target web pages with the accuracy of 87%. The experiments show that techniques like traffic morphing and randomized pipelining can impose high costs, but are unable to stop the WF attack. This attack is able to ignore packet sizes, while most of the attacks on Tor work based on packet sizes.

A problem with the defense techniques that use packet padding or traffic morphing is that they impose extra overhead to the network, in terms of both delay and band width. This issue often makes them unpractical and inefficient for

anonymizing networks. In [26], authors proposed a defense that mitigates this problem. The idea proposed there is much closer to our work; the original approach is that, instead of inserting dummy packets into the traffic, adding the packets that are predicted to be needed by the user in a short time. Of course in that way also some extra packets are added to the traffic changing the fingerprint. In addition, [26] provides a mathematical model that gives the user some hints to estimate the cost of using the defense. To predict which packets must be injected into the traffic, [26] makes use of link pre-fetching and web caching. The difference between their work and our approach is that they are still trying just to inject extra packets to the traffic, while our final goal is more than that. First, we intend to have some experimental results to see how pre-fetching affects the WF attack. In addition, we propose a new defensive mechanism based on changing the amount of pre-fetched data from time to time. In other words, [26] only injects pre-fetched data to change the traffic pattern. Therefore, it could be possible to train a classifier for the pre-fetched enabled streams as well. In contrast, we present a set of experiments that show how we can fight the WF attack by altering the amount of pre-fetched data, which will be discussed in more detail later.

3.3 Criticism

In the literature, some work always debate over the practical feasibility of the WF attack. This section tries to summarize issues questioning the practicality of WF.

There are many different factors that play a role in the success of WF. The main criticism is that academic papers usually oversimplify WF attack models by making some unrealistic assumptions over users' browsing habits, training and testing traces, even the version of Tor used for testing/ training. In [16], it is discussed that the studies performed in [9, 14, 20, 24, 22] simplify the problem and overestimate the adversary's capabilities.

Another criticism is that all works lunch attacks on individual pages, instead of overall websites. So, although the attack is called website fingerprinting, it is actually about webpage fingerprinting. In addition, there are some main factors, including the the hypothesis state space and the size of the instance space, that affect the accuracy of a classifier. Even, the number of training samples provided to the classifier and false positive rates matters. Since reliable feature information is constant, with the increase in the number of classification categories, the classifier eventually runs out of descriptive feature information, which causes either true positive accuracy goes down or the false positive rate goes up. [2] discusses that the effects of such factors are quite observable in the papers with a sufficient world size.

Although more attention should be paid to the scenarios by which attacks are evaluated, we should not dismiss WF as a threat. However, [2] argues that, due to theoretical and practical issues, realistic WF attacks are hard to lunch on Tor. Therefore, it claims that even simple defenses could protect Tor users against WF. It seems that the Tor community believes that defense mechanisms do not need to be very complicated to be effective.

To the best of our knowledge, none of the defense techniques has investigated the effect of link pre-fecting on WF. In the next section, we will describe the concept of link prefetching. Since most of the defenses are applied at the Tor network, they are required to be acceptable by the Tor community. However, we aim to propose a defense technique which can also be used by the website owners.

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 [8]. 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 [11, 12]. 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 [19]. 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 some variations for different types of prefetching called DNS prefetching also, (specified as `<link rel="dns-prefetch" ..>`) which is supported by *Mozilla Firefox* and *Google Chrome*. Another form of expression `<link rel="prerender" ..>` also does the same job as `prefetch` in *Google Chrome* and *Microsoft Internet Explorer*.

Figure 3 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 hovers his mouse cursor on it. However, it can be seen on the network timeline (on the right bottom) that the image is pre-fetched right after loading the page, not when the user actually requested it. 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 hovering 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 An Example of Packet Sequences

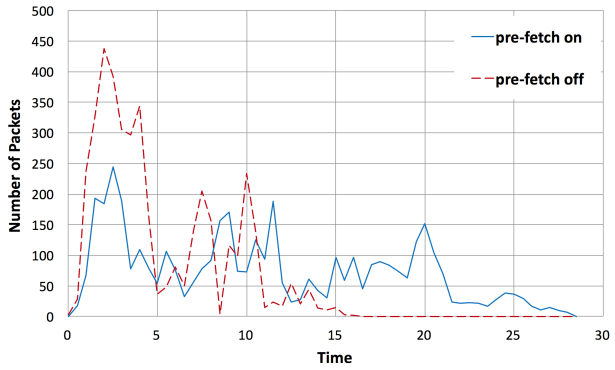

Figure 4: A website fingerprint for pre-fetch on/off cases

Figure 4 illustrates a website fingerprint in terms of interpacket timing, where the *x-axis* corresponds to 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 incoming and outgoing packets combined are 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 as shown in the figure that the packets for the *on* case are more scattered throughout time compared to the pre-fetch-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 would be able to characterize how the *fingerprint* of a specific website looks like. Please note also that interpacket 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 a set of website of interest, for a set of features that attacker determines to use. In this subsection, we summarize the definition of the four most popular features defined by Cai et al. [8] to discuss the effect of prefetching on this features in the following subsection.

Packet Sequence: A packet sequence P can be written as $P = \langle(t_1, l_1), (t_2, l_2), ..., (t_n, l_n)\rangle$ [8], where t_i is the interpacket time between packets i and $i-1$, and l_i is the byte length of the i-th packet. We write P_t and P_l as the sequences of only the interpacket times and lengths, respectively.

Unique Packet Lengths: Unique packet length is differnet when there exists a unique packet length L in a sequence of packet lengths P_l such that it belongs to a sequence while it does not to the other. In other words:

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

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: Interpacket timing is different if the packets of the same index from the two different sequences P and P' have different interpacket timing:

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

Based on these features, Cai et al. [8] claimed that if $P \neq P'$, then one of the four features shall differ.

4.4 The Effect of Prefetching on the Features

Based on the features we summarized in Section 4.3, this subsection explain how prefetching can affect the four features. We will compare the two packet sequences P and P' for the same webpage, where P is the packet sequence without prefetching and P' is the packet sequence where prefetching was enabled (and where there is more than one pre-fetch request).

Packet Sequence: By the definition of link prefetching [**?**], pre-fetch is requested only after a webpage is fully loaded and after an idle time. Thus, for a packet sequence P where prefetching was disabled, another packet sequence P' with prefetching enabled would not differ except the prefix of P'. In other words, P' is a concatenation of two sequences P and Q such that $P' = P \| Q$, where Q is the incoming and outgoing packets for pre-fetch request and download such that:

$$Q = \langle(t_{n+1}, l_{n+1}), (t_{n+2}, l_{n+2}), ..., (t_{n+m}, l_{n+m})\rangle, m > 1 \quad (5)$$

Unique Packet Lengths: In order for the *unique packet lengths* to be different between P and P', it is sufficient to have one packet with a different length in P' that do not appear in P, or vice versa. When there are tailing packets Q for P', the likelihood is high that there is one or more packets in Q that do not exist in P. Although in some cases the multiset of packet lengths M_l' can be a subset of M_l, it is not likely considering that the prefetching resources are different ones from the resources that are already fetched – the designer of a website would not want to pre-fetch the same resource that's already been downloaded. Thus, unique packet lengths will be different between P and P'.

Packet Length Frequency: This feature differs when there exists a packet length L such that the frequencies differ between P and P', while appearing at least once in both P and P'. This is highly likely to be different considering that most the packets are transmitted with the

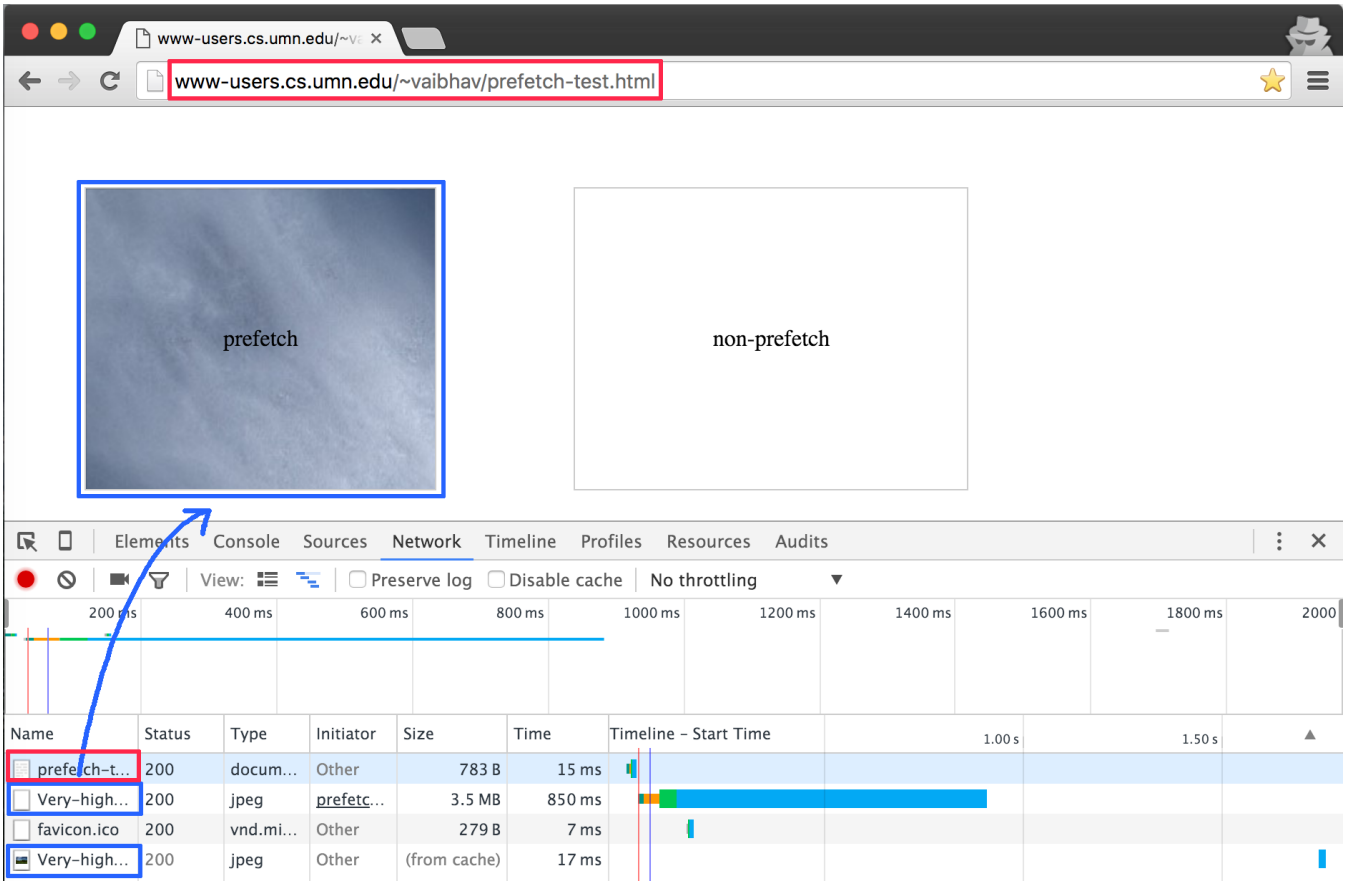

Figure 3: Network timeline showing pre-fetch

size of MTU (maximum transmission unit), which is typically 1500 bytes. When the requested resource is bigger than 1500 bytes, which is true in most cases, the number of packts with length 1500 will be larger in P' than in P ($n_{1500}(P') > n_{1500}(P)$) and thus yield false for this feature.

Packet Ordering: P and P' obviously do not have the same packet ordering because $P_l \neq (P \| Q)_l$.

Interpacket Timing: This feature would not be affected because of the clause $min(|P|, |P'|)$ would always yield $|P|$, and $(P_t)_i$ where i is between 1 and $|P|$ would not be affected by Q.

In summary, at least one out of features is affected by link prefetching. Furthermore, if our assumptions about the prefetching packets are valid, at most three features are affected by link prefetching, which is sufficient for a classifier to put P' in a different class to P.

We show the effect of prefetching in various settings throughout a series of experiments.

5. EXPERIMENTAL EVALUATION

We present the design of two sets of experiments for evaluating the effect of link prefetching on website fingerprinting and the results from these experiments in the following subsections.

5.1 Research Question

Table 1: Experiment design to answer RQ1

train\test	prefetch on	prefetch off
prefetch on	(1)	(2)
prefetch off	(3)	(4)

We formulate the research questions of this project as follows:

RQ1: Does prefetching itself provide an extra degree of defense?

RQ2: Can prefetching be used as a browser-side defense mechanism?

We summarize our plans to tackle *RQ1* in Table 1. The rows represent the link prefetching setting used during the training stage by the attacker and the columns represent the link prefetching setting used during the testing stage by the attacker.

1. We speculate that prefetching itself might provide extra defense because of the extra packets. As per the Link Prefetching FAQ[11], Firefox prefetches web content requested by the webpage at browser idle time where browser idle time is defined as the time when the web page has finished loading. If this is implemented similarly in the Tor browser, prefetched pack-

ets requests and responses should flow into the browser after it has finished downloading resources required by the webpage. It can also be the case however, prefetching websites are more vulnerable to fingerprinting because of the extra prefetch upstream and downstream packets that creates a distinct suffix.

2. This case is unlikely to present itself in reality since the prefetching option is turned on by default in the latest version of the Tor browser as of this writing. 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 the prefetching option off in his/her browser. This scenario, therefore has the potential to confuse the attacker.

4. This case simulates a situation where a victim is loading websites that does not prefetch any resource. This can be thought of the most normal scenario as of now and can be used for checking the strength of our fingerprinting classifier.

For *RQ2*, we found that the performance of link prefetching as a browser-side defense mechanism depends primarily on two parameters.

1. number of prefetching requests made by the browser

2. total size of prefetched responses received by the browser

We vary the values of both these parameters and present our results in Section 5.5.3.

5.2 Experimental Design

We have completed two sets of experiments corresponding to *RQ1* and *RQ2*. Our first experiment consists of checking the accuracy of known classifiers using coarse features for the four scenarios described above. Our second experiment consists of checking the effect of randomly varying the number of prefetching requests and size of prefetched responses on the classifier accuracy.

5.3 Data Collection for RQ1 and RQ2

We created a web scraper using a Python framework called Scrapy[4] and ran on the top 10,000 most popular websites of the top 1 million as listed by Alexa[3]. On scraping through the HTML content of the top 6000 websites for link prefetching keywords such as *prerender*,*next*,*prefetch* we found 60 websites using link prefetching for caching web content in the browser. We then configured a virtual machine to run a Tor browser crawler[1] to visit each of these 60 websites 16 times and captured all packets seen. This was done once with the *prefetch-next* option left to its default value and repeated with this option turned off. By doing this, we captured packets for link prefetching as done currently in the real world by some of the most popular websites.
When doing the data collection for *RQ2*, we uploaded a copy of the Wired homepage[6] and uploaded on our webspace[7]. We then created nine different variations of this page by changing the number of prefetching requests in this page to 10,50,100 and changing the size of prefetched response per request to 100,1 kB, 10 kB. We used the Tor browser crawler[1] to capture packets for each of these webpages 144 times. In order to have equal data for comparison for some other websites, we also captured data for two other websites which we found to be similar.

5.4 Feature Set

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.

5.5 Experimental Validation

5.5.1 Naive Bayes Classifier

5.5.2 Support Vector Machine-based Classification

5.5.3 Results

5.6 Experimental Validation for *RQ2*

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 prefetching affect fingerprinting attacks. Based on our result, we are planing to suggest some defense mechanisms.

8. ACKNOWLEDGMENTS

This report is submitted as a partial requirement for the CSCI5271 research project. The authors acknowledge the guidance of Professor Stephen McCamant in the University of Minnesota throughout the course of this research project.

9. REFERENCES

[1] A crawler based on tor browser and selenium. https://github.com/webfp/tor-browser-crawler, 2015. Accessed: 2015-12-04.

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

[3] Does alexa have a list of its top-ranked websites ? - alexa support. https://support.alexa.com/hc/en-us/articles/200449834-Does-Alexa-have-a-list-of-its-top-ranked-websites-, 2015. Accessed: 2015-12-14.

[4] Scrapy - a fast and powerful scraping and web crawling framework. http://scrapy.org, 2015. Accessed: 2015-12-04.

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

[6] Wired. http://www.wired.com, 2015. Accessed: 2015-12-04.

[7] Wired. http://www-users.cs.umn.edu/~byunx063/wired_prefetch_10_1.html, 2015. Accessed: 2015-12-04.

[8] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *the 2014 ACM SIGSAC Conference*, pages 227–238, New York, New York, USA, 2014. ACM Press.

[9] 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.

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

[11] D. Fisher. Link prefetching FAQ. https://developer.mozilla.org/en-US/docs/Web/HTTP/Link_prefetching_FAQ, 2003.

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

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

[21] M. Perry. Experimental defense for website traffic fingerprinting. https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting, 2011. Accessed: 2015-11-23.

[22] Y. Shi and K. Matsuura. Fingerprinting attack on the tor anonymity system. In *Information and Communications Security*, pages 425–438. Springer, 2009.

[23] 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.

[24] 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.

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

[26] S. Yu, T. Thapngam, S. Wei, and W. Zhou. Efficient web browsing with perfect anonymity using page prefetching. In *Algorithms and Architectures for Parallel Processing*, pages 1–12. Springer, 2010.