Depending on HTTP/2 for Privacy? Good Luck!

Gargi Mitra
Department of CSE
IIT Madras
gargim[at]cse.iitm.ac.in

Prasanna Karthik Vairam
Department of CSE
IIT Madras
pkarthik[at]cse.iitm.ac.in

Patanjali SLPSK
Department of CSE
IIT Madras
slpskp[at]cse.iitm.ac.in

Nitin Chandrachoodan
Department of EE
IIT Madras
nitin[at]ee.iitm.ac.in

Kamakoti V
Department of CSE
IIT Madras
kama[at]cse.iitm.ac.in

***Abstract*—HTTP/2 introduced multi-threaded server operation for performance improvement over HTTP/1.1. Recent works have discovered that multi-threaded operation results in *multiplexed* object transmission, that can also have an unanticipated positive effect on TLS/SSL privacy. In fact, these works go on to design privacy schemes that rely heavily on multiplexing to obfuscate the sizes of the objects based on which the attackers inferred sensitive information. Orthogonal to these works, we examine if the privacy offered by such schemes work in practice.**

In this work, we show that it is possible for a network adversary with modest capabilities to completely break the privacy offered by the schemes that leverage HTTP/2 multiplexing. Our adversary works based on the following intuition: *restricting only one HTTP/2 object to be in the server queue at any point of time will eliminate multiplexing of that object and any privacy benefit thereof*. In our scheme, we begin by studying if (1) packet delays, (2) network jitter, (3) bandwidth limitation, and (4) targeted packet drops have an impact on the number of HTTP/2 objects processed by the server at an instant of time. Based on these insights, we design our adversary that forces the server to serialize object transmissions, thereby completing the attack. Our adversary was able to break the privacy of a real-world HTTP/2 website 90% of the time, the code for which will be released. To the best of our knowledge, this is the first privacy attack on HTTP/2.

***Index Terms*—HTTP/2 attack, HTTP/2 privacy, encrypted traffic analysis**

I. Introduction

The Facebook-Cambridge Analytica data scandal [1] showed the monetary value of seemingly unimportant personal preferences (e.g., Facebook pages liked) of website users. In recent times, a more severe form of privacy attack, that does not even require access to the end user or the server devices, has been reported. This form of attack is carried out by an adversary that compromises a network device that sits between the client and the server and relies merely on the encrypted traffic exchanged between the two. Further, this attack is totally non-intrusive and does not require compromising the decryption key. Instead, the adversary analyzes the encrypted traffic using statistical and machine learning techniques to reveal *web access patterns* that are indicative of user preferences. Many research works [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12] have highlighted the vulnerability of HTTP/1.x (with SSL/TLS) based webservers to these encrypted traffic analysis attacks.

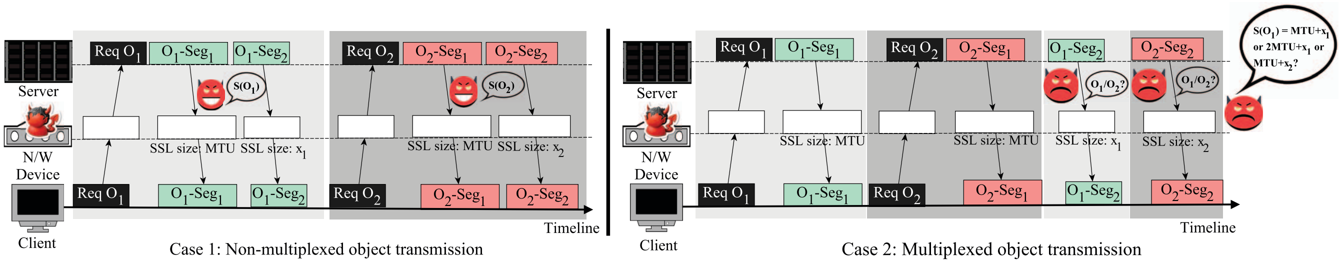
Traffic analysis attacks on HTTP/1.x-based websites use the sizes of encrypted objects (images, JavaScript files, etc.) transmitted by the server to uniquely identify the webpage accessed by the client. For instance, a research work [13] was able to identify the political leaning of users of a popular survey website [14, 15, 16] just based on the size of encrypted image files. Although there exist other channels through which such information can be inferred by an adversary, encrypted object size is the most reliable and widely used side-channel. Consequently, the defense techniques [17, 18, 19, 20, 21] against such attacks aim to obfuscate object sizes, albeit at the cost of unreasonable CPU and bandwidth overheads, which often make them impracticable.

The HTTP/2 protocol [22] is being seen by recent research works [23, 24] as an efficient alternative to such expensive defense techniques. The idea is to use the newly introduced multi-threading feature of the HTTP/2 server that can potentially multiplex multiple objects during transmission. By multiplexing the objects, the resulting TCP stream will have interleaved segments belonging to different objects, making it difficult for a network eavesdropper to discern the encrypted sizes of objects of interest. Therefore, the parallelism in object transmission, although introduced for performance improvements, has the potential to improve privacy without any additional effort. Orthogonal to these efforts, in this paper, we investigate the following question:*'Can we depend on HTTP/2 Multiplexing for obfuscating encrypted object sizes?'*.

We show that a network adversary with the same privileges as a traditional on-path passive eavesdropper can completely break the privacy guaranteed by HTTP/2 Multiplexing. Our adversary leverages the simple fact that spacing consecutive GET requests sent by a client will eliminate any opportunity for the server to multiplex the objects corresponding to these requests. We begin by constructing an adversary that alters network parameters such as latency, jitter, bandwidth, and packet drops. We then observe the effect of each parameter on the multiplexing of objects within an HTTP/2 stream. Our adversary achieves inter-request spacing by adding jitter at the compromised network device immediately after the request corresponding to the object of interest is forwarded. Doing so results in non-multiplexed transmission of the targeted object, thereby breaking the privacy. However, adding jitter has the side-effect of intensified multiplexing for the subsequent objects, which would pose a challenge for the adversary if it were interested in those objects also. Note that the intensified multiplexing is mainly due to the retransmission requests sent for the delayed object (due to jitter) by the client's TCP layer.

To suppress the intensity of multiplexing of subsequent objects, we use the following strategies: **(1)** limit network

978-1-7281-5809-9/20/$31.00 ©2020 IEEE
DOI 10.1109/DSN48063.2020.00044



Case 1: Non-multiplexed object transmission

Case 2: Multiplexed object transmission

Fig. 1: Estimating object sizes from encrypted traffic in non-multiplexed (i.e., O_2 after O_1) vs. multiplexed object transmissions (i.e., $O_1Seg_1 \rightarrow O_2Seg_1 \rightarrow O_1Seg_2 \rightarrow O_2Seg_2$)

bandwidth, and **(2)** reset the ongoing HTTP/2 streams. The adversary limits the network bandwidth to ensure that a limited number of client requests reach the server. This reduced the scope for multiplexing at the server, albeit not completely. When this is not sufficient, the adversary may choose to force the server to reset all the ongoing HTTP/2 streams. Doing so will restart the transmission for the subsequent objects with a clean slate. To achieve this, our adversary drops the packets carrying retransmitted objects, which in turn makes the client time-out and request a server reset. Following this, the client again requests for the subsequent object of interest. At this point, the adversary repeats the attack as it did for the first object of interest. We demonstrate the effectiveness of our enhanced adversary using the website www.isidewith.com, a political survey website that was recently migrated from HTTP/1.1 to HTTP/2 for privacy reasons. Our attack was able to discern all objects of interest 90% of the time. To the best of our knowledge, this is the first traffic analysis attack on HTTP/2 technology.

The rest of the paper is organized as follows: Section II covers the background necessary to understand the paper. Section III provides a high-level overview of our attack. Section IV provides a comprehensive analysis of the impact of network parameters on HTTP/2 multiplexing. Section V describes our adversary and evaluates its impact on a real-world website. We discuss the literature related to our work in Section VI. Finally, we conclude the paper and provide future directions in Section VII.

II. Background

HTTP/2 Multiplexing. The advent of HTTP/2 protocol brought significant improvements over HTTP/1.1 in terms of user perception of latency during webpage loads. Contrary to HTTP/1.1, where requests are processed sequentially, HTTP/2 allows concurrent server threads to serve multiple objects on the same TCP connection, effectively *multiplexing* them. Such a design is particularly useful in avoiding Head-of-Line blocking – the scenario where a large object in the queue blocks the subsequent objects from being processed. This phenomenon is prevalent in HTTP/1.1 and has been widely exploited by adversaries for performing traffic analysis.

The multiplexed object transmission in HTTP/2 makes it difficult for a passive adversary to identify individual objects when TLS/SSL is used. Figure 1 illustrates how a passive network adversary perceives non-multiplexed and multiplexed object transmissions. Case 1 shows a simple scenario where the client requests for O_2 after the transmission of O_1 is completed. The network adversary that has access only to the size of the packets can still determine that the transmission of O_1 is complete the moment it receives a delimiting packet, i.e., the last packet with size that is less than (rarely equal to) the *Maximum Transmission Unit* (MTU). Similarly, it can determine the start and end of transmission of O_2. Consequently, the adversary can sum up the packet sizes corresponding to O_1 and O_2 to determine their sizes, from which the objects' identities can be revealed. On the other hand, in Case 2, the client requests for O_2 before the transmission of O_1 is completed. Consequently, the segments of O_1 get interleaved with those of O_2, thereby making it difficult for the adversary to estimate the sizes of the individual objects. This fact has been used by recent works to thwart privacy attacks that depend on the encrypted object sizes.

Privacy of HTTPS Communication. Privacy, in the context of Internet communications, is a very fluid concept. In this work, we consider *privacy* as the property of a client-server communication due to which the fine details of the client's browsing activities are protected from a passive network eavesdropper. An example of such information is the webpage(s) visited by the client within a website. Despite the use of TLS/SSL encryption, prior works have shown that it is possible to infer these fine details by analysis of encrypted network traffic. A comprehensive study of such works suggest that, for accurately identifying from encrypted traffic if a particular webpage within a website was accessed, it is sufficient if **(1)** the webpage contains at least one object which is not contained in any other webpage in the website, **(2)** the size of the object is different from those of all other objects that constitute the website, and **(3)** the eavesdropper is able to infer the encrypted object size from the network trace. Hence, we consider that the privacy of a webpage is completely broken if a network adversary is able to infer the size of an object that uniquely identifies the webpage (provided it exists) from encrypted traffic. So far, researchers have been successful in launching privacy attacks on HTTP/1.x. On the other hand, the privacy of HTTP/2 in the aforementioned context is still an ongoing topic that is being investigated. There are early indications that HTTP/2 inherently prevents such privacy attacks since estimating object sizes from encrypted traffic is not possible due to multiplexing [23, 24]. Investigating the privacy of HTTP/2 is the primary objective of our paper.

The privacy of the communication channel can also benefit from orthogonal technologies such as ToR. However, a larger proportion of Internet users still do not have access to such services and are therefore, susceptible to traffic analysis based attacks. Even when using ToR, traffic analysis could be employed at the weakest spots such as the ToR entry guard. Therefore, we strongly advocate a comprehensive investigation of the privacy offered by the underlying application layer protocols such as HTTP/1.1 or HTTP/2, before designing defense techniques that supplement them.

Network Parameters. We have seen that multiplexing can prevent attempts by a passive network adversary to estimate object sizes. However, we suspect that multiplexing can be affected by variations in network conditions – they may influence the TCP layer parameters such as congestion window size, which may, in turn, affect the HTTP/2 parameters such as the intensity of multiplexing. We consider the following network variations in our work:

1) ***Delay.*** An addition of a uniform delay in the network increases the Round Trip Time (RTT) of every packet exchanged between the client and the server. However, the time interval between the transmission of two consecutive packets on the network remains the same as in the case of the network with no delay.
2) ***Jitter.*** Most real-world networks experience jitter, i.e., variable network delays. Contrary to a network with uniform delay, in a jittery network, the inter-packet time intervals are not consistent, when compared to those in case of the network without jitter.
3) ***Bandwidth throttling.*** Bandwidth throttling is often performed by Internet Service Providers (ISPs) to limit the number of packets transiting the network. Bandwidth limitation may result in slow loading of images and videos in webpages and degraded quality of video streaming.
4) ***Targeted Packet drops.*** Packets can be dropped in a targeted manner from any intermediate network device, either due to benign reasons (such as congestion control at a router) or due to an adversary residing on the router.

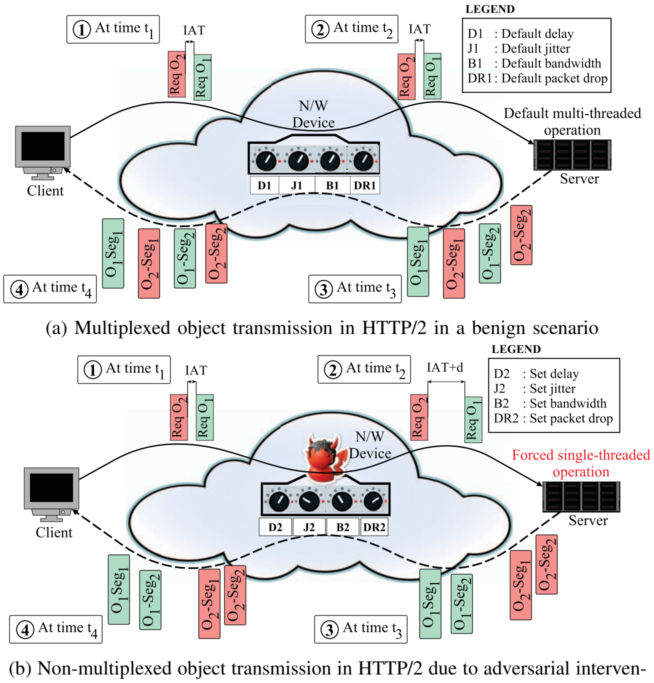
As we will see, changes in traffic patterns at the network layer affect the multiplexing behavior of the HTTP/2 server. The implications of this finding are particularly alarming since even a passive network adversary can control network conditions to a certain extent.

A. Measuring Privacy

We define a metric to quantify the extent of multiplexing of objects, which, in turn, will help us in determining if the information targeted by the adversary is private or not.

Degree of Multiplexing. We define the *degree of multiplexing* of an object as the fraction of bytes of the object that is interleaved with those of another object within the same TCP stream.

Implication on Privacy. From the definition of degree of multiplexing, we can say that, in the scenario where the adversary is able to reduce the degree of multiplexing of an object to *zero*, i.e., no byte of the object is interleaved with another object, they can estimate the encrypted object size. For instance, if two specific resources are required to uniquely identify a webpage, a scheme that prevents the multiplexing of the two objects is sufficient to break the privacy.



(a) Multiplexed object transmission in HTTP/2 in a benign scenario

(b) Non-multiplexed object transmission in HTTP/2 due to adversarial intervention

Fig. 2: High level overview of proposed adversarial scheme

III. High Level Overview

In this section, we provide a high-level overview of our adversarial scheme. The objective of our adversary is to infer the webpages accessed by a targeted client (or user) within a targeted HTTP/2 website using encrypted network traffic analysis techniques. Our adversary uses encrypted object sizes as a side-channel. Using encrypted object sizes in HTTP/2 is particularly challenging due to the multiplexed nature of object transmission. Our adversary is designed to overcome this challenge.

Figure 2 shows the capability of our adversary and its impact on HTTP/2 multiplexing. We assume that the adversary knows which objects to focus on, for breaking user privacy. Hence, it is sufficient for the adversary to prevent the multiplexing of only these *objects of interest*, which we assume to be O_1 in our example. Figure 2a depicts the scenario without our adversary. At time t_1, the client sends two successive GET requests for O_1 and O_2. Let IAT be the time interval between the two requests, which is very small. At time t_2, the packets are routed through the intermediate network device and reach the server. At time t_3, the segments of O_1 and O_2

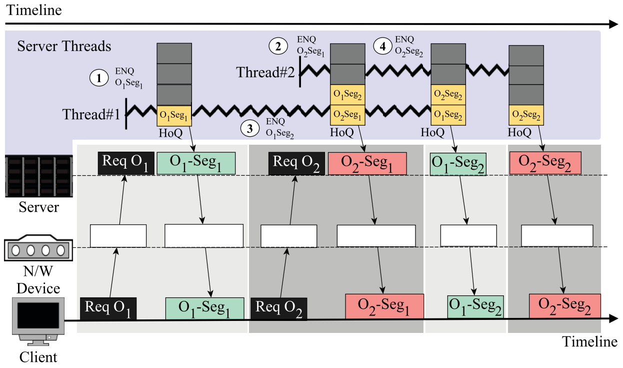


Fig. 3: Interleaved object transmission by an HTTP/2 server

are transmitted in an interleaved manner due to server multi-threading. Finally, at time t_4, the object segments reach the client in an interleaved manner, where they are assembled by the browser.

Figure 2b depicts the same scenario with our adversary controlling an intermediate networking device. Our adversary changes the network parameters to delay the GET request for O_2 by an additional time d (with respect to arrival time of O_1). This extra delay in arrival of O_2's request ensures that the server completes transmitting O_1 before initiating the transmission of O_2. Thus, the network adversary prevents the multiplexing of the objects of interest. When the adversary is interested in more than one object in the same TCP stream, the attack gets challenging due to the repercussions of the previously introduced delay. We describe how our adversary overcomes these in the rest of the paper.

Capability of Adversary. Our adversary is a compromised network device on the client-server path that can **(1)** access unencrypted header fields of both control and data packets, **(2)** monitor size of encrypted packets, **(3)** delay packets, **(4)** limit the bandwidth of the transit link, and **(5)** drop packets. At the outset, this adversary model might seem stronger than the popular passive adversary model used in prior works [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 23, 24], which would only capture network traffic in promiscuous mode. However, in reality, the privilege required (i.e., superuser access) for doing so is sufficient for tuning the network parameters. Such an adversary model is not only realistic but also common today [25]. Also, since we assume a non-intrusive adversary, MITM attacks that involve key hijacking and/or server impersonation, or require compromising the client's browser are out of scope of this work.

Assumptions and Scope. We make the following assumptions for our adversarial scheme. **(1)** We assume that the goal of our adversary is to infer the identity of objects from the encrypted HTTP/2 stream. Once this is done, any of the techniques from the HTTP/1.x literature can be used to launch a full-fledged privacy attack. **(2)** We assume that the adversary does not have the capability to decrypt at the point of real user traffic collection. Breaking encryption is indeed a difficult problem to solve, and there exists no practical work that can do this. This assumption holds true for most authorized and unauthorized parties except maybe those involving government agencies. **(3)** We assume that the adversary's behavior cannot be detected easily at the client or the server since it mimics network issues that are commonplace. **(4)** We also assume that the adversary has sufficient time ($\approx$ few minutes) to access the website and tune the network parameters before launching the attack. **(5)** We assume that the adversary has the knowledge about the sequence in which objects in the webpage are requested for and the specific object that is of interest.

IV. Impact of Network Parameters on HTTP/2 Multiplexing

In this section, we explain how the network parameters affect multiplexing at the server. We start by extending Figure 2 to show the server side operations. Figure 3 shows the threads that get created at the server corresponding to the GET requests sent by the client. The server, on receiving the first request, spawns a thread ($Thread\#1$), which starts enqueueing the first segment of O_1 on the server queue for transmission on the TCP stream, as shown in Step ①. However, before $Thread\#1$ could enqueue the second segment of O_1, (i.e., $O_1 - Seg_2$), the server receives a request for the object O_2 from the client and spawns a second thread ($Thread\#2$) that starts enqueueing segments of O_2 on the server queue (refer Step ②). This is followed by the enqueueing of $O_1 - Seg_2$ (refer to Step ④) and $O_2 - Seg_2$ (refer to Step ⑤) respectively. In the presence of TLS/SSL encryption, this interleaving of object segments makes it difficult for our network adversary to estimate object sizes, as discussed in Section III.

We reinforce our explanations with experimental results on an object in the webpage showing the results for the '2020 Presidential Quiz' in the website *www.isidewith.com*. The object of interest is an HTML file of size ≈ 9500 bytes. By default, the degree of multiplexing of this object is $\approx 98\%$. We now examine the change in the degree of multiplexing of this object with change in each of the network parameters, viz., delay, jitter, bandwidth, and packet drops.

A. Delay

Introducing uniform delay for all packets on client → server path in the network cannot increase the inter-arrival time between two successive packets at the server. Hence, we do not use this parameter for our adversarial scheme.

B. Jitter

The inter-arrival time between two successive GET requests sent by a client can be increased by delaying each of the request packets by an unequal amount of time. The proposed adversary can achieve this by introducing a calculated amount of network jitter on the client-server path. For instance, in our example, the first request can be delayed by 0 ms, second by d ms, the third by $2d$ ms, and so on, to achieve an inter-arrival spacing of d ms. The amount of jitter to be introduced should depend on the size of the object of interest, the time elapsed since the previous GET request, and the time interval

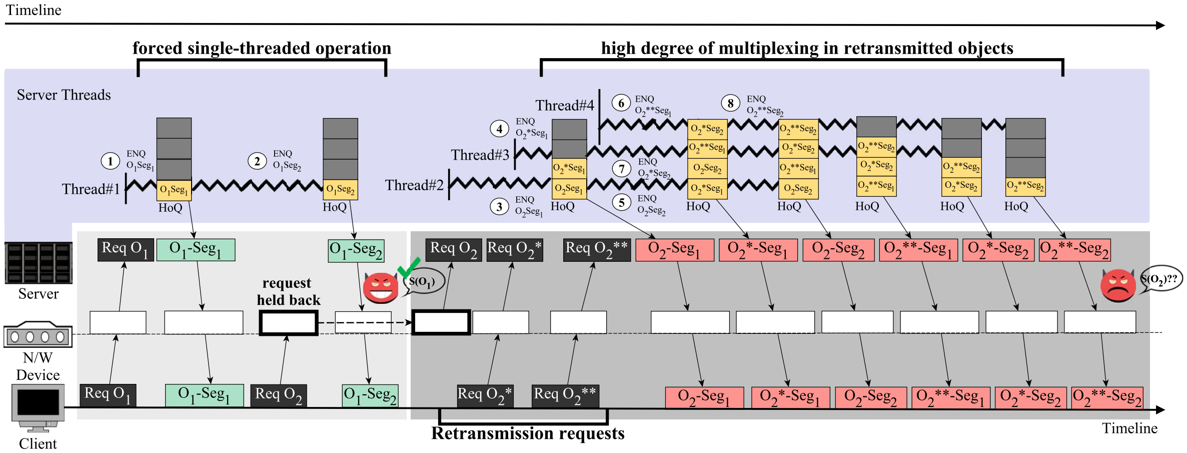


Fig. 4: Effect of network jitter on multiplexing of objects in HTTP/2

before the issuance of the next GET request by the client under normal network conditions.

Figure 4 illustrates the HTTP/2 server behavior under jittery network conditions by extending Figure 3. The adversary holds back the client request for O_2 at the intermediate network device for a longer time than usual. Consequently, the server thread serving O_1 (i.e., $Thread\#1$) completes the transmission of all the segments of O_1 in single-threaded mode before the request for O_2 reaches the server (refer to Step ① and Step ②). Therefore, if O_1 is the only object of interest to the adversary, then its start and end points of transmission and hence its size can be estimated.

When the adversary is also interested in a subsequent object (O_2 in this example), introducing jitter in the network can introduce complications. This happens when the client request for O_2 is delayed for a considerably long period of time. This may lead the server to send duplicate acknowledgements (dup-ACKs) to a previous GET request sent by the client, which makes the client believe that the request for O_2 has been dropped in the channel. When this happens, the client sends a bunch of retransmission requests, called *TCP Fast-retransmits*, for the same object to the server within a short span of time (denoted by $ReqO_2{}^*$ and $ReqO_2{}^{**}$). The retransmitted requests cause the HTTP/2 server to spawn multiple concurrent threads ($Thread\#2$, $Thread\#3$ and $Thread\#4$) that serve multiple copies of O_2 to the client over the same TCP stream in interleaved manner, thereby resulting in intensified multiplexing.

In our experiment, the object of interest in *isidewith.com* is the 6^{th} object downloaded by the client. Table I shows the impact of increasing jitter values on the degree of multiplexing of the object. For each jitter value shown in the table, the webpage was downloaded 100 times. Initially the number of cases where the object was not at all multiplexed increased steadily with an increase in the amount of jitter until 50 ms. However, increasing jitter further did not have any effect. This was because, at this point, the number of retransmission requests (> 130) started increasing manifold and the segments of the retransmitted objects got interleaved with those of the object of interest. Therefore, we set the jitter such that the inter-arrival time of requests is 50 ms. The next challenge for our adversary is to reduce the number of object retransmissions and object retransmission requests.

Increase in delay per request (ms)	Cases where object of interest was not multiplexed (%)	Increase in no. of retransmissions (%)
0 (baseline)	32	0 (baseline)
25	46	≈ 33
50	54	≈ 130
100	54	≈ 194

TABLE I: Effect of jitter on HTTP/2 multiplexing

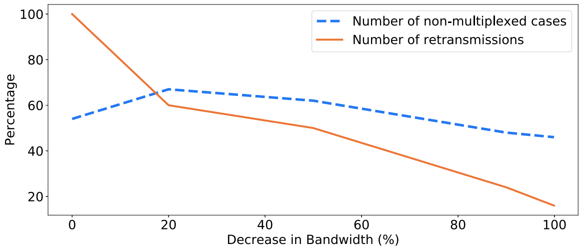


Fig. 5: Effect of bandwidth limitation on multiplexing of objects in HTTP/2

C. Bandwidth Limitation

Reducing the bandwidth during an ongoing communication session, given a particular amount of delay in the network, reduces the bandwidth-delay product (BDP). BDP is a measure of how much information (outstanding packets) the communication channel can hold at a given instant of time. When the BDP reduces, the TCP protocol at the communication endpoints responds to this change by decreasing the size of the TCP sender window. We rely on this intuition to reduce the number of fast-retransmit requests sent by the client.

We extend the setup described in Section IV-B with bandwidth limits applied at the compromised network device. Note that the bandwidth limits are applied for both incoming and outgoing packets. Our experiment starts with a channel band-

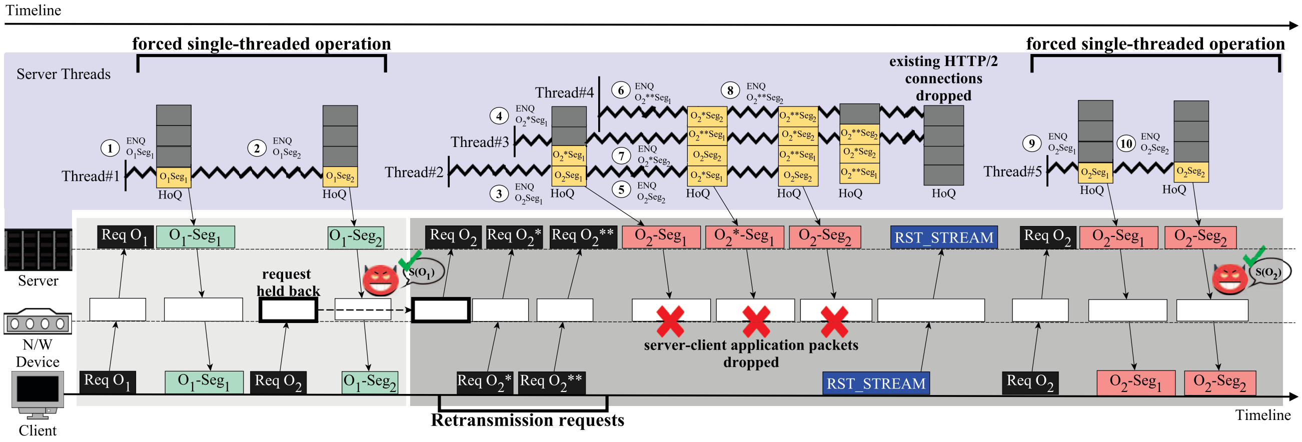


Fig. 6: Targeted packet drop by adversary to force non-multiplexed object transmission in HTTP/2

width of 1000 Mbps. During the course of the experiment, we varied the bandwidth to 800 Mbps, 500 Mbps, 100 Mbps and 1 Mbps. For each bandwidth, the webpage was downloaded 100 times and we noted the corresponding degree of multiplexing of the object.

Our experimental results show that throttling the network bandwidth indeed reduces the number of retransmissions as seen in Figure 5 (solid line). Contrary to our expectation, the percentage of cases in which our object of interest was in non-multiplexed state, i.e., success cases (dashed line), initially goes up sharply till 800 Mbps but reduces gradually right after. When we delved deeper, we understood that a significant percentage of successes observed till 800 Mbps can be attributed to a retransmitted version of the object and not the actual object. With a rapid decline in the number of retransmissions post 800 Mbps, the number of such success cases also reduce, thereby affecting the overall number of success cases at the lower bandwidths. Note that it was not possible to reduce the bandwidth beyond 1 Mbps because that resulted in a broken connection. Based on our observations, we limit the bandwidth of the medium to 800 Mbps. However, this can only improve the success rate to a small extent.

D. Targeted Packet Drops – forcing HTTP/2 Stream Reset

In this section, we investigate if another feature of HTTP/2 protocol, namely, the HTTP/2 stream reset, can further improve the success rate of our attack. The HTTP/2 reset stream was introduced to enhance the efficiency of a server especially when dealing with a highly lossy communication channel. Typically, the client sends the TCP fast-retransmit requests to deal with network layer delays and losses that are not persistent. However, when the communication channel is highly lossy, the client sends the HTTP/2 reset stream signal to the server (indicated by a packet with the corresponding HTTP/2 stream number and RST_STREAM flag set). Reset Stream is particularly interesting since the server closes the stream and flushes the corresponding object segments from its queue thereby reducing the load on the network immediately. In response to the lossy medium, the client's TCP stack also increases the timeout for fast-retransmits. After Stream Reset, the client resends GET requests if a high priority object is not yet received. We now exploit this feature to improve the success rate of our attack.

Figure 6 depicts a scenario that builds on the scenario described in Figure 4: firstly adding jitter (refer section IV-B) followed by limiting the network bandwidth (refer section IV-C). To force a HTTP/2 stream reset, our adversary drops many packets on the server → client path (mimicking a lossy network) carrying the objects segments until the client resets the streams as shown in Figure 6. After reset, the client again sends the request for O_2. The client's TCP also waits for a longer time before attempting to send fast-retransmission requests. Consequently, by this time, the server can completely transmit O_2 in single-threaded mode. The most challenging part of this phase of the attack is to identify when to start and stop dropping packets. This is because the adversary cannot discern the retransmitted objects from the actual ones. The adversary can either use a timer or use the number of forwarded GET requests (identified by using the filter *'ssl.record.content_type==23'*).

We experimentally verified the effect of targeted packet drop (with both jitter and bandwidth limits applied) on HTTP/2 multiplexing on the HTML file of size 9500 Bytes. Since the object is the 6^{th} in the stream, the adversary drops 80% packets on server → client path from the time of transmission of the 6^{th} GET request from the client. We continue the packet drops for 6 seconds until the client sends stream reset. The experiment was repeated 100 times and we observed a success rate of ≈ 90%, where the object of interest was transmitted in non-multiplexed state after the *Reset Stream* signal was sent by the client. However, further increasing the packet drop rate resulted in a broken connection.

V. Our Privacy Attack

We now demonstrate a real-life adversarial attack [1] using the insights obtained from Section IV. However, finding a

[1] Repository: https://bitbucket.org/gmit91/http2_privacy_attack

real website with the complete implementation of HTTP/2 protocols is difficult today. Most HTTP/2 websites do not enable multiplexing either due to lack of awareness (disabled by default) or to avoid the extra storage required. Since our objective was to *disprove* that HTTP/2 multiplexing offers privacy, we demonstrated on the strongest HTTP/2 website known to us.

We chose *isidewith.com* [14], since it is the most widely cited website in the fingerprinting literature. This website allows users to identify their political leanings for the *'2020 Presidential Elections'*. It is also noteworthy that this is the same website that migrated from HTTP/1.1 to HTTP/2 shortly after privacy concerns were raised in a prior work [13]. Our evaluation suggests that currently it is resilient to most of the known attacks.

Target Website. The website works as follows. In response to the survey, a webpage containing the list of 8 political parties in order of preference of the user is displayed. The webpage consists of an HTML page that contains hyperlinks of 47 embedded objects, such as JavaScript files, images, and style files. One of these JavaScripts, on execution, makes the client to consecutively send request to the server for 8 images within a short span of time. Each image corresponds to the emblem of one of the 8 political parties, and the order in which the client issues requests for these images is the same as that in which they are displayed on the result webpage. The images of the political party emblems are of size ranging between $5KB$ to $16KB$. Our adversary attempts to infer the order in which the 8 political parties appear on the webpage by encrypted traffic analysis. In short, the adversary has 9 different objects of interest – one HTML file and 8 image files. Note that our adversary has a pre-compiled list of image size to political party mapping which it leverages to complete the attack.

Client setup. The experiment was conducted with the help of several volunteers (≈ 500) participating in the survey from different client systems inside our laboratory over a period of three months. They were asked to use the Firefox browser, which, to the best of our knowledge, is the only browser to have implemented HTTP/2 multiplexing efficiently. Each time, the volunteers noted down the survey results (sequence of political parties), which was later used as the golden reference to determine the prediction accuracy of our adversary.

Adversary Setup. We designated our lab's gateway (with 1 Gbps link) as the adversary. The gateway monitored the transiting traffic and whenever it could detect a connection to isidewith server, it launched the attack. The adversary was made up of 3 basic components – (a) the traffic monitor, which was implemented using tshark; (b) the network controller, which was implemented using bash scripts; and, (c) the object prediction module, which was implemented using Python scripts. In the first phase of the attack, the adversary introduced jitter (of 50 ms additional delay) in the client-server communication path and also started counting the number of GET requests in the client→server path. As soon as the client sent the 6^{th} GET request (that corresponds to the HTML file), the adversary reduced the bandwidth to 800 Mbps and simultaneously started dropping 80% application packets in the server→client path. It does so for the next 6 seconds to force the client to send a Reset Stream signal to the server. After this point, the jitter value was increased to 80 ms additional delay per GET request packet so as to force the server to transmit the 8 consecutive image files in non-multiplexed form.

Object (O_{curr})	$HTML$	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8
T(Req O_{curr}) $-$ T(Req O_{prev}) (ms)	500	780	0.4	2	0.3	0.1	0.3	2	0.5
T(Req O_{next}) $-$ T(Req O_{curr}) (ms)	160	0.4	2	0.3	0.1	0.3	2	0.5	26
Sucess (%) of Adversary Target: One object at a time	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
Sucess (%) of Adversary Target: All objects at a time	**90**	**90**	**85**	**81**	**80**	**62**	**64**	**78**	**64**

TABLE II: Prediction Accuracy: O_{prev} and O_{next} denote the objects requested by the client immediately before and after O_{curr} respectively. T(Req X) denotes the time of request for object X by the client.

Results. Table II shows the accuracy of our adversary in identifying the 9 different objects of interest from the encrypted traffic. The 8 images can appear in any order in the survey result, depending on the user response, where I_i is the i^{th} image in the sequence. In absence of any adversarial intervention, the degree of multiplexing of each of these objects range from 80% to 99%. We consider our attack to be successful only when the adversary is able to bring down the degree of multiplexing of the object of interest to 0% and identify it from the encrypted traffic. In the context of *isidewith.com*, when the adversary is interested in just one object, our adversary was able to find it with 100% accuracy all the time as shown in Table II. On the other hand, if the adversary were to be interested in identifying the entire sequence of images, the accuracy was highest for I_1 but gradually drops for later images. This is because the later images required adding more jitter which resulted in a broken connection. Additional jitter was required since the previous, current, and next GET requests are issued by the client in quick succession. For instance, request for I_5 has spacing of 0.1 ms and 0.3 ms with the previous and and next requests as shown in Table II.

Ethical Considerations. Adherence to ethical standards was strictly ensured during all experiments. All the volunteers were made aware about the goal of our research work and they had absolutely no stake in the outcome of the survey. We also informed the developers of $isidewith.com$ about our findings and the need to fix their website.

VI. Related Work

Ours is the first research work to perform encrypted traffic analysis on HTTP/2. Prior works [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12] in this domain target users of HTTP/1.x websites to predict the websites/webpages they accessed, either over anonymous communication channels such as Tor [26] or over simple encrypted channels such as HTTPS. Recent research works [23, 24] have ascertained the difficulty of extending the attack techniques used in HTTP/1.x to HTTP/2. Specifically, the researchers highlight the complications introduced by HTTP/2 multiplexing in the context of object size based attacks. While it may seem intuitive to use machine learning or even deep learning to learn the complex patterns in HTTP/2 traffic, our paper uses an alternative approach. Our work focuses on preventing multiplexing from occurring in the first place, thereby simplifying the attack.

The adversarial scheme presented in this paper affirms the fact that HTTP/2 multiplexing is not a reliable and sound enough feature for designing defense solutions against website fingerprinting attacks.

VII. Conclusion and Future Work

Privacy schemes that rely on HTTP/2 multiplexing are not dependable and can be attacked by our proposed adversary. Our adversary is built on the general principles stated in the paper and can be extended to other real-world websites/scenarios. Our contribution is especially significant since HTTP/2 is in the early stages of getting adopted worldwide, and communicating the pitfalls of recent research works on its privacy is important. Going forward, we identify the following research directions.

Improving the Adversary. The success rate of our adversary drops to 60% in cases where a large number of consecutive objects are of interest. We believe that triggering the packet drops and jitter addition accurately will alleviate this problem, possibly using machine learning. Another possible extension would be to infer the object identity even when the object is partly multiplexed. Our preliminary experiments suggest that this is indeed possible, however, at the cost of employing complex analysis techniques. In our limited experience, we have observed that variations in network speed may result in innumerable ways in which the objects can be multiplexed, all of which need to be accounted for.

Exploring other types of web traffic Exploring the suitability of our technique for other types of web traffic, such as streaming traffic, is an interesting direction. We strongly believe that our attack technique can supplement the existing attacks on HTTP/2 streaming [27], which is also in its nascent stages.

Improving HTTP/2 privacy. Several HTTP/2 features such as server push and prioritization that are not a function of the underlying network can be leveraged for privacy. For instance, the client can opt for a different priority/order of object delivery every time, thereby confusing the adversary.

VIII. Acknowledgement

We would like to thank all the anonymous reviewers and our shepherd, Professor Rüdiger Kapitza, for their valuable comments which helped us improve the quality of the paper.

References

[1] (2020) Facebook–Cambridge Analytica data scandal. Last accessed: 16 Mar 2020. [Online]. Available: https://en.wikipedia.org/wiki/Facebook-Cambridge_Analytica_data_scandal

[2] H. Cheng and R. Avnur, "Traffic analysis of ssl encrypted web browsing," *http://www.cs.berkeley.edu/˜daw/teaching/cs261-f98/projects/final-reports/ronathan-heyning.ps*, 1998.

[3] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu, "Statistical identification of encrypted web browsing traffic," in *Proceedings of IEEE Symposium on Security and Privacy*, 2002, p. 19.

[4] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-channel leaks in web applications: A reality today, a challenge tomorrow," in *Proceedings of IEEE Symposium on Security and Privacy*, 2010, pp. 191–206.

[5] K. Zhang, Z. Li, R. Wang, X. Wang, and S. Chen, "Sidebuster: automated detection and quantification of side-channel leaks in web application development," in *Proceedings of ACM CCS*, 2010, pp. 595–606.

[6] G. Danezis, "Traffic Analysis of the HTTP Protocol over TLS," 2010. [Online]. Available: http://www0.cs.ucl.ac.uk/staff/G.Danezis/papers/TLSanon.pdf

[7] P. Chapman and D. Evans, "Automated black-box detection of side-channel vulnerabilities in web applications," in *Proceedings of ACM CCS*, 2011, pp. 263–274.

[8] X. Gong, N. Borisov, N. Kiyavash, and N. Schear, "Website detection using remote traffic analysis," in *Proceedings of PETS*, 2012, pp. 58–78.

[9] B. Miller, L. Huang, A. D. Joseph, and J. D. Tygar, "I know why you went to the clinic: Risks and realization of https traffic analysis," in *Proceedings of PETS*, 2014.

[10] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt, "A critical evaluation of website fingerprinting attacks," in *Proceedings of ACM CCS*, 2014, pp. 263–274.

[11] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website Fingerprinting at Internet Scale," in *Proceedings of NDSS*, 2016.

[12] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," in *Proceedings of ACM CCS*, 2018, pp. 1928–1943.

[13] Lescisin, Michael and Mahmoud, Qusay, "Tools for Active and Passive Network Side-Channel Detection for Web Applications," in *Proceedings of $USENIX$ Workshop on Offensive Technologies ($WOOT$)*, 2018.

[14] (2020) ISideWith. Last accessed: 16 Mar 2020. [Online]. Available: https://www.isidewith.com/

[15] Thousands of people have changed their votes after taking this quiz. Will you? Last accessed: 16 Mar 2020. [Online]. Available: https://www.washingtonpost.com/news/the-intersect/wp/2016/11/07/thousands-of-people-have-changed-their-votes-after-taking-this-quiz-will-you/

[16] Web Quiz Tells You Which Presidential Candidate Best Fits Your Worldview. Last accessed: 16 Mar 2020. [Online]. Available: https://www.npr.org/sections/itsallpolitics/2012/07/19/157048377/web-quiz-tells-you-which-presidential-candidate-best-fits-your-worldview

[17] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis," in *Proceedings of NDSS Symposium*, vol. 9, 2017.

[18] 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 *Proceedings of NDSS*, vol. 11, 2011.

[19] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg, "A systematic approach to developing and evaluating website fingerprinting defenses," in *Proceedings of ACM CCS*, 2014, pp. 227–238.

[20] W. Cui, J. Yu, Y. Gong, and E. Chan-Tin, "Realistic Cover Traffic to Mitigate Website Fingerprinting Attacks," in *Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 1579–1584.

[21] K. Al-Naami, A. El Ghamry, M. S. Islam, L. Khan, B. M. Thuraisingham, K. W. Hamlen, M. Alrahmawy, and M. Rashad, "BiMorphing: A Bi-Directional Bursting Defense Against Website Fingerprinting Attacks," *Proceedings of IEEE Transactions on Dependable and Secure Computing*, 2019.

[22] Hypertext Transfer Protocol Version 2 (HTTP/2). Last accessed: 16 Mar 2020. [Online]. Available: https://tools.ietf.org/html/rfc7540

[23] R. Morla, "Effect of Pipelining and Multiplexing in Estimating HTTP/2.0 Web Object Sizes," in *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1707.00641

[24] W. Lin, S. Reddy, and N. Borisov, "Measuring the Impact of HTTP/2 and Server Push on Web Fingerprinting," *Workshop on Measurements, Attacks, and Defenses for the Web*, 2019.

[25] (2020) Shodan. Last accessed: 16 Mar 2020. [Online]. Available: https://www.shodan.io/

[26] (2020) Tor. Last accessed: 16 Mar 2020. [Online]. Available: https://www.torproject.org/

[27] H. Wu, G. Cheng, and X. Hu, "Inferring ADU Combinations from Encrypted QUIC Stream," in *Proceedings of International Conference on Future Internet Technologies*, 2019, pp. 1–6.