Snoopy: A Webpage Fingerprinting Framework With Finite Query Model for Mass-Surveillance

Gargi Mitra, Prasanna Karthik Vairam, Sandip Saha, Nitin Chandrachoodan, *Member, IEEE*, and V. Kamakoti

Abstract—Internet users are vulnerable to privacy attacks despite the use of encryption. Webpage fingerprinting, an attack that analyzes encrypted traffic, can identify the webpages visited by a user. The key challenges in performing mass-scale webpage fingerprinting arise from (i) the sheer number of combinations of user behavior and preferences to account for, and; (ii) the bound on the number of website queries imposed by the defense mechanisms (e.g., DDoS defense) deployed at the website. These constraints preclude the use of conventional data-intensive ML-based techniques. In this work, we propose Snoopy, a first-of-its-kind framework, that performs webpage fingerprinting for a large number of users visiting a website. Snoopy caters to the generalization requirements of mass-surveillance while complying with a bound on the number of website accesses (finite query model) for traffic sample collection. We show that Snoopy achieves $\approx 90\%$ accuracy when evaluated on most websites, across various browsing contexts. A simple ensemble of Snoopy and an ML-based technique achieves $\approx 97\%$ accuracy while adhering to the finite query model, in cases when Snoopy alone does not perform well.

Index Terms—Encrypted traffic analysis, mass surveillance, website privacy, webpage fingerprinting

1 Introduction

Leakage of private information is one of the biggest concerns for Internet users today. Recent reports [1], [2] suggest that sensitive information that could cause imminent personal harm to Internet users, including banking passwords, salary details, health records, location information, and CCTV footage, have been leaked in the dark web. While privacy loss for personal information is easily perceivable, it is not so obvious for other types of information. For example, the Cambridge Analytica scandal showed that the political leanings of an ordinary individual may not be worthy to the attackers, but the political alignment of a larger demography can help them predict the outcome of an election [3]. Such incidents show that some user information that are seemingly unimportant to an individual might inadvertently turn sensitive when collected on a *mass scale*.

Mass Surveillance and its Consequences. One of the largest sources of mass-scale information about personal preferences of Internet users are their web browsing activities [4], [5]. Information such as the *identity of websites* visited by users from a demography could be useful to an attacker, for instance, to gauge the popularity of websites in the region. However, more fine-grained information such as the *identity of webpages* (a *website* can have many *webpages*) visited by the users on a targeted website could be much more useful to the attackers. For instance, a surge in the number of visitors to the “fixed-deposit” page of a bank website shortly after the announcement of a new policy is critical to the bank. This information, if leaked, could help competing banks estimate its growth and also design counter-marketing strategies. Another example is when the adversary identifies the most popular webpage of the website for placing malware. Such active attacks (phishing attacks [6], drive-by-download attacks [7] and denial-of-service attacks [8]) cause financial loss either to the hosting organization or its consumers or both. In this paper, we focus on designing a *mass-surveillance method* that can potentially reveal such *fine-grained* sensitive information from a large number of Internet users based on their *web browsing activities*.

Straightforward methods to identify the webpages visited by the users of a website include compromising the end-user devices and the websites' servers (e.g., extracting decryption key, installing fake certificates). However, these methods are highly intrusive and are easily detectable, making them unsuitable for long-term surveillance. Note that the importance of information grows manifold when collected over a longer period of time [9], [10], [11], [12], since it ensures sufficient coverage of users and also reflects changing trends in their behavior. Encrypted Traffic Analysis (ETA) [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25] (specifically, webpage fingerprinting) is the most promising non-intrusive method to collect such information for long periods of time by merely capturing the encrypted traffic exchanged between the websites' server and the end-users. The webpages are identified by

- *Gargi Mitra, Sandip Saha, Nitin Chandrachoodan, and V. Kamakoti are with the Indian Institute of Technology Madras, Chennai, Tamil Nadu 600036, India. E-mail: {gargim, CS20S044, kama}@cse.iitm.ac.in, nitin@ee.iitm.ac.in.*
- *Prasanna Karthik Vairam is with the National University of Singapore, Singapore 119077. E-mail: prasanna@comp.nus.edu.sg.*

Manuscript received 23 March 2022; revised 5 November 2022; accepted 9 November 2022. Date of publication 16 November 2022; date of current version 1 September 2023.
(Corresponding author: Gargi Mitra.)
This article has supplementary downloadable material available at https://doi.org/10.1109/TDSC.2022.3222462, provided by the authors.
Digital Object Identifier no. 10.1109/TDSC.2022.3222462

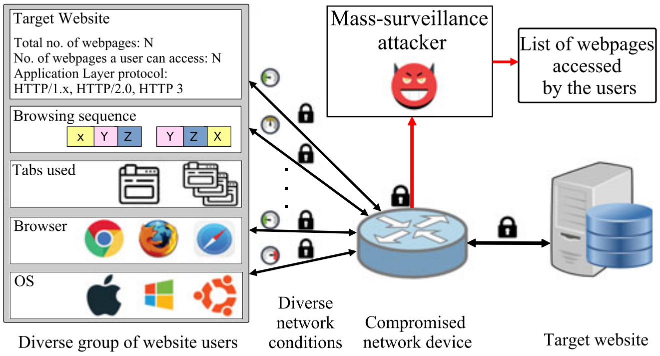

Fig. 1. Snoopy, the proposed framework for webpage identification attack on a mass-scale.

formulating signatures that can identify them uniquely when accessed by users through encrypted channels. While several existing webpage fingerprinting attacks [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25] target particular users, our goal is to perform mass-surveillance by observing the activities of a *large number of users* visiting a particular website. Fig. 1 shows a high-level objective of this paper. The main challenge with mass-surveillance is accounting for a diverse set of user behavior (e.g., number of tabs used), user preferences (e.g., OS/Browser used), and network conditions. We collectively refer to these factors as the *browsing context* in this paper.

Existing techniques [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25] for webpage fingerprinting are not suitable for mass-scale traffic analysis even with just HTTPS (i.e., without Tor/VPN). Practical mass-surveillance requires a *generalized model* that can identify webpages irrespective of the *browsing context* of a user. Building a generalized model that accounts for all possible browsing contexts is challenging. The primary hindrance in the generalization of existing works is the massive number of data points required, i.e., the number of traffic samples required per webpage to account for all possible browsing contexts. For instance, building a generalized model for 3 operating systems and 3 browsers will require 9 times the traffic samples as compared to one-OS-one-browser scenario [24]. Moreover, for achieving the accuracy promised by DL/ML algorithms, collecting adequate number of traffic samples per scenario is also crucial. Existing ML-based [24] and DL-based techniques [26], [27] that aim at generalization of fingerprinting models have demonstrated the significance of a large number of data points in learning traffic features and building webpage fingerprints accurately. Additionally, any change in the website also requires fresh collection of traffic samples and re-training the model [28]. Therefore, generalizing existing techniques requires assuming a query model that allows the adversary innumerable accesses to the website within a short span of time. However, in practice, we observed that the DDoS/DoS mitigation schemes at the websites detect and block IP addresses that make such attempts. Furthermore, IP addresses that cause anomalies in website analytics, including those involved in attempts to sample webpages repeatedly, are identified and blocked. Note that, an adversary can neither keep collecting traffic samples from a website over a long period of time to bypass these anomaly detection systems. This is because, the structure and contents of websites change significantly and frequently. For instance, the American financial services website B_5 (refer to Table 3) used in our experiments changed (added/deleted/modified) 65 of its webpages over a period of only 7 days. This problem has also been pointed out in a prior research work [28].

To work around the problem of collecting a massive number of traffic samples, prior works have assumed knowledge about one or more of the following, thereby compromising generalization:

1) *User interests* – Existing works [13], [14], [15], [16], [23] assume that the user is only interested in a subset of webpages. In mass-surveillance, however, the adversary needs to account for a wide variety of user interests;
2) *User behavior* – Most of the existing works [15], [17], [20] assume restricted user behavior such as single-tab browsing [15], [24], [25] and traversing only a limited number of webpages per session [23]. A few works account for the use of more than one tab [17], [20], but they were not successful beyond 2 browser tabs. However, such assumptions are not valid in practical scenarios. Accounting for multi-tab browsing requires webpage-sequence based data collection, which in turn increases the traffic samples exponentially. Further, during multi-tab browsing, a user might even load two webpages in parallel. Existing studies [29], [30] have discussed the parallel multi-tab browsing behavior of Internet users in further details;
3) *User preference* – Existing works [16], [18], [25] assume prior knowledge about the OS and the browser used, as well as knowledge about browser caching (on/off) and cookie values. However, mass-surveillance necessitates accounting for various combinations of OS and browser configurations; and,
4) *Network conditions* – Existing works build prediction models for a particular network condition, which cannot be generalized to other network conditions. However, mass surveillance requires the prediction model to work across geographical locations and service providers.

Naïve attempts at generalization without the addition of adequate number of data points for covering additional scenarios result in poor prediction accuracy. For instance, we witnessed a drastic drop in prediction accuracy (e.g., from 96% to 76% in case of Wfin [21]) of existing ML-based works when we broadened the user interest from a small number of webpages[1] to a large number (refer to Fig. 7 in Section 6.4.3). Further, we also noticed a drop in prediction accuracy (e.g., from 78% to 58% in case of ML_OPS [14]) when the number of training samples is reduced from 10 to 3 per webpage (refer to Fig. 6 in Section 6.4.2). Likewise, when we tried to generalize existing works in terms of user preference (caching on or off), we found a drop in accuracy (as shown in Table 5 in Section 6.4.4) when a model built for one scenario (caching on) was used to predict webpages accessed from a different scenario (caching off). Therefore,

1. We conducted these experiments on a popular bank website.

availability of a limited number of training samples imposes restrictions on the use of existing DL/ML based techniques.

In our work, we propose *Snoopy*, a practical webpage fingerprinting framework, that meets the generalization requirements of mass-surveillance, while assuming a *finite query model*. With limited number of traffic samples, accounting for the numerous scenarios encountered in mass-surveillance is challenging. Snoopy uses domain knowledge about the transport and application protocols to collect traffic samples in a focused manner. For instance, Snoopy uses *encrypted web resource size*, a simple feature used in ETA, that remains largely unaffected by changes in network conditions, eliminating the need to cover this scenario altogether. For generalizing across other scenarios Snoopy relies on static analysis of web object sizes, HTML code and headers to estimate the expected fingerprints. In this paper, we analyze the capability of a simple model such as ours in terms of generalization in cases where there are practical constraints for using ML/DL techniques. We show that Snoopy was able to achieve more than 90% accuracy for most of the websites we considered, when tested on traffic samples from a diverse set of browsing contexts. For the few websites where Snoopy achieved comparatively lower accuracy ($\approx 80\%$), we show that it is possible to improve the accuracy to as high as 97% by using an ensemble of Snoopy and an ML-based technique, that complies with the constraints of a finite query model. To the best of our knowledge, ours is the first work to attempt webpage fingerprinting attack for mass surveillance. We intend to release our code and artefacts[2] to interested researchers.

The rest of the paper is organized as follows. In Section 2 we present the literature related to our work and motivate the need for Snoopy. Next, we state our adversarial capabilities and assumptions, followed by a high level overview of Snoopy in Section 3. Then we describe the design choices made for Snoopy in Section 4. This is followed by a detailed description of Snoopy in Section 5. Thereafter, we describe the implementation and evaluation strategies of Snoopy and present a detailed analysis of the experimental results in Section 6. We discuss the consequences of mass-surveillance and suggest some possible countermeasures to the proposed technique in Section 7. Finally, we conclude the paper in Section 8.

2 Related Work

With growing security concerns, traffic on the Internet is encrypted. Thus, our work is related to the literature on Encrypted Traffic Analysis (ETA) techniques, which can be broadly classified into (1) Website identification attacks – these works [33], [34], [35], [36], [37], [38], [39] aim to infer coarse-grained web browsing information, i.e., the identity of websites browsed over Tor/VPN; and, (2) Webpage identification attacks – these works [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [31], [32] aim to infer fine-grained web browsing information of users from HTTPS traffic, i.e., the identity of webpages browsed within a website. The techniques used for webpage identification are very different from those used for website identification. On one hand, website identification require features (e.g., Round trip time) that vary across websites but remain consistent for webpages within a website. In contrast, webpage identification attacks [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [31], [32] require features (e.g., HTML size) that vary across the webpages within a website, which can help distinguish between them. Blurring the lines between the two classes, some website identification attacks [34], [35], [36], [37], [38], [39] internally use webpage fingerprinting techniques to distinguish between the homepages of thousands of websites. We note that such techniques do not inherit the same set of challenges that a webpage identification attack faces (e.g., restrictions on number of accesses to the website). In this work, we focus on webpage identification attacks on a given website, subject to constraints on the number of webpage accesses.

In this section, we revisit the literature on webpage fingerprinting and assess their suitability for mass-scale HTTPS traffic analysis. Webpage fingerprinting on mass-scale traffic requires a generalized classifier which could account for diversity in client behavior and network conditions. Diversity in client behavior, for example, in terms of browser/OS/device used, number of tabs open, and the sequence in which webpages are accessed, results in different fingerprints for the same webpage [22]. Diversity in network conditions in terms of jitter, bandwidth and packet drop rates also affects the webpage fingerprints [35]. Accounting for these diversities across so many factors is challenging due to practical restrictions on the number of website accesses. In this context, Table 1 classifies the prior works in terms of their generalization capabilities. The earliest webpage fingerprinting techniques [31], [32] could successfully generalize webpage fingerprints for simple contemporary websites that did not use caching, cookies, and used static webpages with limited number of resources. Such works used basic features such as size of client request packets and sequences of encrypted object sizes, coupled with elementary statistical methods.

The prevalence of web caching on the Internet prompted subsequent research works [13], [14], [15] to start the use of webpage-sequence fingerprinting to account for variations in traffic patterns based on the previously accessed webpage(s). However, the applicability of webpage-sequence fingerprinting techniques is restricted only to targeted attacks and are not suitable for mass surveillance due to the following reasons – (1) It necessitates an enormous number of website accesses as compared to individual webpage based fingerprinting techniques, even for medium-sized websites; and, (2) these works used features such as resource ordering [13] and traffic burst patterns [14], [15] for fingerprinting webpage sequences. Such features are not consistent across varying network conditions and user behavior (e.g., number of browser tabs open), and hence, require an estimate of the network speed and browsing behavior of the victim. Such fingerprinting techniques were useful in the context of surveillance on a small set of targeted users, since the attackers had knowledge about the targeted victim's browsing behavior and network conditions. However, assuming interests of website users, their browsing behavior (sequential or single-tab browsing) and

2. Link to repository: https://gmit91@bitbucket.org/gmit91/snoopy.git

TABLE 1
Compliance of Webpage Fingerprinting Techniques With the Requirements for Mass-Surveillance

Existing works	Mass-surveillance Requirements					
	Generalization Requirements					Compliance with a finite query model
	Caching	Cookies	Network conditions	User interests	Multi-tab browsing	
Cheng and Avnur [31], Sun et al. [32]	✗	✗	✓	✓	✗	✓
Cai et al. [15], G. Danezis [13], Chapman and Evans [14], Gong et al. [33]	✓	✗	✗	✗	✗	✓
Miller et al. [16], Hayes and Danezis [18]	✓	✓	✗	✗	✗	✓
Gu et al. [17], Zhuo et al. [34]	✓	✓	✗	✓	✓	✗
Xu et al. [20]	✗	✗	✗	✗	✓	✗
Alan and Kaur [22]	✗	✓	✓	✗	✗	✓
Ghiette et al. [23]	✓	✓	✗	✓	✗	✗
Panchenko et al. [19], Sirinam et al. [35]	✓	✓	✓	✓	✗	✗
Wang et al. [25]	✗	✓	✗	✓	✗	✓
Shen et al. [24]	✗	✗	✗	✓	✗	✓
Snoopy	✓	✓	✓	✓	✓	✓

their network conditions is impractical for mass-surveillance in realistic scenarios [28].

The use of cookies by the next generation websites further complicated the process of webpage fingerprinting, even for targeted attacks. Tracking cookies embedded inside web resources and session cookies included in application layer headers result in resource size variations. To account for these variations, subsequent research works [16], [18] either used complex techniques (combination of clustering algorithms, Gaussian distribution, Hidden Markov Model and Viterbi algorithm) complemented by huge datasets [16] or simplified their assumptions about user behavior [18]. The former technique [16] is too restrictive about user interests since they consider only a small set of webpages to make data collection practical. On the other hand, the latter one [18] assumes impractical user behavior such as single tab browsing without caching. However, these techniques are restricted only to targeted attacks, and are not suitable for mass surveillance due to the following reasons – (1) The goal of mass-surveillance is to understand the interests of the users of a website. Therefore, restrictive assumptions about user interests are unreasonable in the context of mass-surveillance, and; (2) Restrictive assumptions regarding the browsing behavior (for e.g., the number of tabs used, OS/ Browser used, and the browser configuration) of a diverse set of website users are also unreasonable. Therefore, these techniques cannot be used for analysis of encrypted traffic on a mass scale.

Recent works [17], [19], [20], [22], [23], [24], [25], [34], [35] on webpage fingerprinting have recognized the importance of practical mass surveillance. Most of these works acknowledge the necessity of accounting for diversity in user interests [19], [23], [25], user behavior (particularly, a wide variety of user interests and multi-tab browsing) [17], [20], [34], OS/ browser settings [22] and network conditions [35] while performing encrypted traffic analysis on mass-scale web traffic. However, it is to be noted that each of these works can only generalize for at most one of these factors. This is because, for most of these works [22], [34], [35], generalizing for even one of these factors using the features and/or the techniques presented in these works requires collecting a massive amount of traffic samples, coupled with a heavyweight ML algorithm. This would result in a large number of website queries (i.e., the number of website accesses done by the adversary), as well as a high bootstrap time every time the website contents change or a new OS or browser or firmware becomes popular. For instance, one of these works [22] account for diversity in browser, OS and device used by the client at the cost of an immensely high bootstrap time.

As summarized in Table 1, existing ETA techniques have complied with a finite query model at the cost of restrictive assumptions about the interests and browsing behavior of the users, and vice-versa, and are suitable for mass surveillance. We propose *Snoopy*, an ETA technique that is designed with the goal of meeting the generalization requirements and complies with a finite query model, both of which are essential for mass-surveillance.

3 The Snoopy Framework

In this section, we present *Snoopy*, our proposed adversarial framework for mass-surveillance through webpage fingerprinting We first describe the capabilities of the adversary, then define the scope of our work, and finally provide a high level overview of Snoopy.

3.1 Adversary Capabilities

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; and, (2) observe the traffic characteristics, such as the size of encrypted packets. Such an adversary model is not only realistic but also common today. In the real world, this translates to a rogue Internet Service Provider (ISP) or a malicious entity who compromises an ISP router.

3.2 Assumptions and Scope

We make the following assumptions about the context in which Snoopy is to be employed.

- We do not consider websites that host dynamic or highly personalized content for each user, for e.g.,

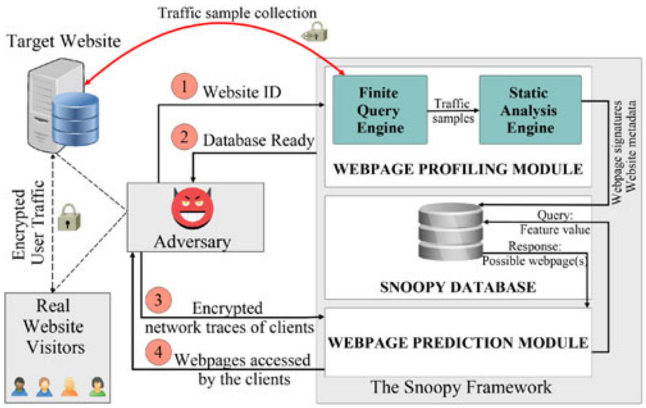

Fig. 2. The components of Snoopy.

social media websites and web search engines, as targets for our attack;

- The adversary does not have the capability to decrypt the traffic of real website users. Most real-world adversaries cannot decrypt web traffic except maybe those involving government agencies or authorized middle-boxes [40]; and,
- Snoopy must compromise a network device that has access to all the encrypted Application layer packets (referred to as *traffic trace*) exchanged between the client and the server for the entire duration of a browsing session. For instance, our framework cannot be used when route flapping (dynamic change of route due to, for instance, unreliable connections) occurs on the server-client path.
- The scope of our work does not include inferring the intent and interest of the website visitors from the webpages inferred by Snoopy. Rather, we believe that existing works [41] on behavioral analysis of website users can be used by an adversary for this purpose.

3.3 High Level Overview

We now provide a high level overview of Snoopy – our mass-surveillance framework. Snoopy comprises the *Snoopy Database* and two functional modules, namely, *Webpage Profiling Module* and *Webpage Prediction Module*. The Snoopy Database is meant to store information (i.e., webpage fingerprints and additional website metadata) required for carrying out the predictions, and the two functional modules are used by the adversary for profiling webpages and predicting the identities of webpages accessed by the users. Fig. 2 shows the different components of Snoopy and how an adversary interacts with them.

To populate the Snoopy Database, the adversary first triggers the Webpage Profiling Module of Snoopy with the Website ID (IP address or homepage URL) of the target website, as shown in Step 1 of Fig. 2. The goal of this module is to gather information about the website that will be useful to the Webpage Prediction Module. The input to this module is the website identifier (IP address or homepage URL) and the set of features to be used for webpage fingerprinting. First, the finite query engine in this module performs a focused traffic sample collection from the target website. Subsequently, this traffic is decrypted and both the encrypted and decrypted versions of the traffic samples are passed on to the static analysis engine. While Snoopy does not require the decryption keys of users during the attack, it still needs to decrypt the traffic samples that it generates on its own during the profiling stage. This is necessary to identify the plaintext resources (e.g., HTML, javascript, images) of each webpage from their encrypted counterparts. The final output of the Webpage Profiling module includes information pertaining to (1) the structure of the website; (2) resource download sequences for each webpage in the website; (3) signature of these web resources; and, (4) other relevant website metadata, such as cache-ability and cookie information of the web resources.

Once the database has been populated, Snoopy notifies the adversary that it is ready to be used for prediction, as shown in Step 2 of Fig. 2. When the adversary wants to predict the webpages accessed by a user from their encrypted traffic trace, it triggers the *Webpage Prediction Module* of Snoopy. This module takes an encrypted traffic trace T of a user as input, as shown in Step 3 of Fig. 2, and predicts the webpages that are accessed in T using the information stored in the Snoopy database. Snoopy performs this prediction in two steps. First, it extracts the feature values from the input trace T and performs a lookup on the Snoopy Database to retrieve the sequence of candidate web resources present in T. Next, Snoopy uses this sequence and the website metadata to predict the final set of webpages, which it returns to the adversary (Step 4 of Fig. 2).

The most unique and crucial feature of Snoopy is its ability to comply with a finite query model, while retaining its generalization capability at the same time. Achieving this balance between generalization and a finite number of queries is extremely challenging. Snoopy attempts to solve this problem using the following:

1) **Predictable features**: It uses a feature that exhibits negligible or predictable variation across browsing contexts. The predictability of the feature values allows Snoopy to fingerprint webpages in different browsing contexts even with a finite number of traffic samples, complying with a finite query model. The feature used in Snoopy and the justification behind this choice are discussed in further details in Section 4; and,
2) **Focused data collection and feature value estimation**: For collecting traffic samples from the targeted website, Snoopy uses a focused data collection method to account for certain browsing contexts, which has been discussed in details in Section 5.1. Snoopy estimates the variations in feature values in different browsing contexts using static analysis of HTTP headers and payload. At the time of prediction, Snoopy uses these estimated feature values (from the Snoopy Database) based on the user's browsing context. The feature value estimation and prediction techniques used by Snoopy are detailed in Section 5.2.

4 Snoopy Design: Predictability of Fingerprints

Snoopy needs an encrypted traffic feature that either exhibits no variation or predictable variation across different browsing contexts, in order to minimize the number of traffic samples required during webpage profiling. In other words, the traffic features used for webpage profiling need to be *stable* across different browsing contexts. For this, we evaluated different classes of traffic features (e.g., timing side-channels and traffic burst patterns) that are widely used in prior ML-based works [16], [33] for targeted attacks and assessed their stability. It is known and also shown in our evaluations presented in the Appendix, available online, that such traffic features have poor stability and are therefore not predictable. For instance, the traffic burst pattern corresponding to a webpage download changes with variations in network conditions and the number of parallel browser tabs used by a user.

Snoopy uses *sequence of encrypted web resource sizes* as the fingerprint of a webpage. The size of an encrypted resource is computed as the sum of TLS segment sizes of all packets carrying the resource. Since each user might have a different browsing context, Snoopy first focuses on *understanding* the effect of various browsing contexts on the fingerprint. While some of the factors of the browsing context affect the encrypted resource size, some others affect the sequence in which the resources are downloaded. An analysis of these factors helps Snoopy *statically* estimate the variations caused by each of these factors, thereby eliminating the need for collecting traffic samples for all possible browsing contexts. We now explain how the size of encrypted resources and their download sequence change based on the browsing context of a user:

1) ***Operating system (OS)*** – The choice of OS affects the TLS segment size of packets in multiple ways. First, the TLS segment size of a web resource depends on the TLS implementation on the corresponding OS. Different OS'es start with the same TLS record but they break them into segments of different sizes. Although one might not expect this to affect the TLS segment size, we noticed minor differences when the sum of TLS segment sizes is calculated. A deeper analysis revealed that the number and size of segments affects the metadata associated with every TLS segment in a number of ways, which are, to our advantage, predictable. First, there are some TLS headers added to every TLS segment. Therefore, if a record is broken into several small TLS segments, the total number of TLS header bytes for all the segments would be greater than the total number of TLS header bytes added in case the record was broken into fewer larger segments. Second, a variable field on the HTTP header, namely, the user agent string that carries the name of the OS also affects the TLS record size.
2) ***Browser*** – The browser name indirectly affects the TLS segment size since it is also a part of the User Agent string. As different browser names have different lengths, they affect the length of the User Agent string, which in turn, results in variation in the TLS record sizes.
3) ***Browsing sequence*** – The sequence in which a user browses the webpages affects both the TLS record sizes and the download sequence of the web resources, particularly when caching and cookies are allowed (enabled in most browsers by default).
   - *Caching:* When a user visits a webpage containing a resource that was previously downloaded, the resource may not get downloaded if resource caching is enabled. This results in variation in resource download sequence. For instance, if a user visits webpage W_X (composed of resources r_1 and r_2), followed by webpage W_Y (composed of resources r_1 and r_3), the overall resource download sequence would be r_1–r_2–r_3. On the other hand, if caching was disabled, it would result in the following sequence: r_1–r_2–r_1–r_3.
   - *Cookies:* When a user allows a website to use cookies, the server sends a *session cookie* in the HTTP header along with the first resource delivered during the browsing session. Addition of the session cookie increases the TLS record size of the first resource downloaded during a browsing session. Therefore, a resource would have a larger TLS record size if it gets downloaded at the beginning of a browsing session, as compared to its TLS record size when it gets downloaded at a later point during the browsing session. Furthermore, another type of cookie, called the *tracking cookie*, affects the payload size of the transmitted resource. Tracking cookies hold information about the browsing behavior of a user such as the URL of the previously browsed webpage(s) in the session. Since different webpage URLs have different lengths, the variation in TLS record size due to tracking cookies depend on the webpage(s) last visited by a user. Also, tracking cookies hold a null value for the first resource delivered during the browsing session. Again, this causes the TLS record size of a resource to vary depending on the user's browsing sequence. For instance, if the size of a resource r_1 is s_1 and the size of the session cookie is sc, then the TLS record size of r_1 would be $(s_1 + sc)$ if r_1 is the first resource to be downloaded. On the other hand, if r_1 is downloaded after the user has visited a webpage of URL length tc (also, the tracking cookie size) in a browsing session, the TLS record size of r_1 would be $(s_1 + tc)$.
4) ***Application layer protocol*** – Grouping packets that carry a resource is critical for computing the encrypted resource size. For HTTP/1.x, packets belonging to a resource have the same TCP ACK number, making the process straightforward. However, it is not the same for HTTP/2 and HTTP/3 websites due to pipelining and multi-threaded server operations. With multi-threading, packets belonging to two different resources could get interleaved within the same TCP stream (in HTTP/2 [42]) or QUIC stream (in HTTP/3 [43]). To handle such complex scenarios, Snoopy adopts the technique described in our recent work [44] for computing encrypted resource sizes in HTTP/2 websites. For HTTP/3 websites, Snoopy drops the QUIC connection establishment packets so that the

communication protocol falls back to HTTP/1.1 or HTTP/2.

5) ***Parallel tabs*** – Browsing on concurrent tabs affects the sequence of downloaded resources. For instance, when a user browses two webpages, say W_X and W_Y one after another on two tabs, the resources of W_X get downloaded first, followed by the resources of W_Y. On the other hand, if the user opens the two webpages in two parallel browser tabs, the resources of W_X and W_Y would get downloaded in an interleaved fashion. In such cases, the attacker faces the additional challenge of identifying the resources download sequence corresponding to each webpage from the interleaved sequence of encrypted resources, which makes the webpage prediction process more challenging.

6) ***Network conditions*** –A congested link results in packet transmission delays, packet drops, and blocking of new connections. Among these, packet drops affect the encrypted size of resources. The dropped packets may or may not get re-transmitted, based on the nature of the resource. In the event of packet re-transmissions, the re-transmitted packets can be easily assembled using state-of-the-art network protocol dissectors so that Snoopy can still work. Therefore, computation of the encrypted resource size by Snoopy is not affected in this case. However, in rare situations when a large number of dropped packets are not re-transmitted or results in a broken connection, Snoopy cannot compute or predict the resource size correctly. Severe network congestion may also result in route flapping. In that case, the attacker would not be able to access the complete network trace unless it has control over the new route too. Such circumstances cannot be handled by Snoopy.

Presently, Snoopy has been configured to use *sequence of encrypted web resource sizes* as the feature for fingerprinting webpages. We incorporate the aforementioned knowledge in the design of Snoopy to make it compliant with a limited query model while allowing it to be generalized at the same time. However, if a feature that is more stable is discovered, Snoopy can be configured to use that instead.

5 Snoopy Design: A Deep Dive

In this section, we provide a detailed description of the two functional modules of Snoopy, viz., the Webpage Profiling Module and the Webpage Prediction Module, and how they use the insights from Section 4. As discussed in Section 4, the present implementation of Snoopy uses only one feature – sequence of encrypted resource sizes. Therefore, in the rest of this section, we describe the functional modules of Snoopy in the context of this feature. Table 2 contains a summary of the notations used in the rest of this section.

TABLE 2
Summary of Notations Used in Section 5

Notation	Meaning
$WebsiteIP$	Website identifier (say, IP address)
EF	Encryption function
F	Feature used for webpage fingerprinting
$\mathbb{G}$	Graph representation of the target website
$\mathbb{W}$	Set of webpages in $\mathbb{G}$. $\mathbb{W}$=$\{w_1, w_2, \ldots, w_n\}$
E	Set of directed edges in $\mathbb{G}$. E= $\{(w_i, w_j) \mid$ *w_j is directly navigable from w_i*$\})$
$\mathbb{RM}$	Webpage to resource map
$\mathbb{R}$	Set of web resources in the targeted website. $\mathbb{R}$= $\{r_1, r_2, \ldots, r_m\}$
$\mathbb{S}$	Set of resource download sequences of all webpages of the target website
$TSamples$	Dataset containing encrypted traffic samples for each r_i
sig_i	Feature value (signature) of the resource r_i
$featureDB$	Web resource signature database.ss
$c_t(r_i, j)$	Increase in size of resource r_i due to tracking cookie when the user navigates from URL j
$c_s(r_i, k)$	Increase in size of resource r_i due to session cookie carrying the Browser/OS identifier bo_k
$c_t(r_i)$	Set of all possible values by which size of r_i might increase due to tracking cookies (See Section 5.1.3)
$c_s(r_i)$	Set of all possible values by which size of r_i might increase due to session cookies (See Section 5.1.3)
$max(c_t)$	Maximum possible increase in size of any resource due to tracking cookies, $max(c_t) = max(c_t(r_i, j)) \forall i, j$
$max(c_s)$	Maximum possible increase in size of any resource due to session cookies, $max(c_s) = max(c_s(r_i, k)) \forall i, k$
$cookie\_var$	Lookup-table containing cookie-induced variations for each resource. Each entry is of the form $< r_i, \{c_t(r_i), c_s(r_i)\} >$
$reverse-FeatureDB$	Reverse database made out of $featureDB$
T	Encrypted traffic trace of a user, to be processed by the Webpage Prediction Model

5.1 Webpage Profiling Module

The input to this module is the target website identifier $WebsiteIP$, the encryption function EF, and the webpage fingerprinting feature F (i.e., encrypted resource size sequence). The output of the webpage profiling module includes the following information (1) the *structure of the website*, represented by a graph $\mathbb{G} = (\mathbb{W}, E)$, (2) a *resource-map* $\mathbb{RM} : \mathbb{W} \longrightarrow \mathbb{R}$ that shows the relationship between webpages in $\mathbb{W}$ and the web resources in $\mathbb{R}$, where $\mathbb{R} = \{r_1, r_2, \ldots, r_m\}$ is the set of web resources that constitute the targeted website ; (3) the *set of resource download sequences* for all n webpages in the website, denoted by $\mathbb{S} = \{S_1, S_2, \ldots, S_n\}$, where S_i be the sequence of resources requested by the client when a user accesses the webpage w_i; (4) *signatures of web resources*, i.e., the encrypted size of each resource; and, (5) other *relevant website metadata*, such as the cache-ability and cookie information of the web resources. We now describe the working of the Webpage Profiling Module using Algorithm 1 in the following sections.

5.1.1 Extracting Structure of the Website

The GET_WEBSITE procedure in Step ① in Algorithm 1 uses automated website-crawling and webpage parsing techniques to construct $\mathbb{G}$, the graph representation of the website of interest. A vertex u in $\mathbb{G}$ has a directed edge to another vertex v if u represents a webpage that has

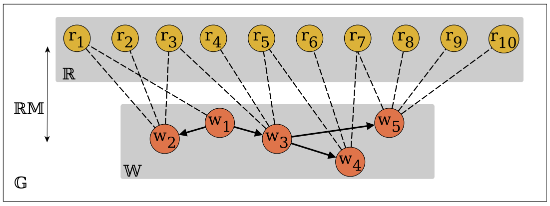

Fig. 3. Representation of the structure of a sample website.

embedded hyperlinks to help users navigate directly to the webpage represented by v. Following this, the GET_RESOURCES procedure in Step ② parses each webpage in $\mathbb{W}$ to extract its resource download sequence and compiles $\mathbb{R}$, the list of embedded web resources present in the website. The set of resource download sequences thus retrieved is denoted by $\mathbb{S}$, and the unique set of embedded resources is denoted by $\mathbb{R}$. The MAP_RESOURCES procedure in Step ③ creates a bipartite graph $\mathbb{RM}$ where the vertices consist of elements in $\mathbb{W}$ and $\mathbb{R}$. A webpage w_i is mapped to a web resource r_j when r_j is embedded in w_i, and this mapping is denoted by an edge between the vertices corresponding to w_i and r_j. Fig. 3 shows an example of $\mathbb{R}$, $\mathbb{W}$, $\mathbb{RM}$, and $\mathbb{G}$ for a website.

Algorithm 1. PROFILE_WEBSITE

input: $WebsiteIP$, EF, F
output: $\mathbb{G}$, $\mathbb{S}$, $\mathbb{RM}$, $reverseFeatureDB$, $cookie\_var$
/* automated website crawling*/
1 $\mathbb{G}$ = GET_WEBSITE($WebsiteIP$);
/ * parsing hypertext * /
2 $\mathbb{S}$, $\mathbb{R}$ = GET_RESOURCES($\mathbb{W}$);
/ * webpage-resource relation * /
3 $\mathbb{RM}$ = MAP_RESOURCES($\mathbb{W}$, $\mathbb{R}$);
/ * traffic sampling for webpage access * /
4 $TSamples$ = COLLECT_SAMPLES($\mathbb{W}$, F);
/ * unique identifiers for encrypted resources * /
5 $featureDB$ = BUILD_SIGNATURES(F, $\mathbb{R}$);
/ * look-up table for prediction * /
6 $reverseFeatureDB$ = CONSTRUCT_DICTIONARY ($featureDB$);
/ * variable cookie fields * /
7 $cookie\_var$ = COMPUTE_COOKIE_VAR($\mathbb{G}$, $featureDB$, EF)

5.1.2 Focused Traffic Sample Collection and Formulating Signatures of Web Resources

For building webpage fingerprints Snoopy requires a dataset comprising traffic samples that capture how different web resources manifest themselves on an encrypted communication channel in different browsing contexts. The COLLECT_SAMPLES procedure in Step ④ of Algorithm 1 builds this dataset. The inputs to this step are the set of webpages $\mathbb{W}$ and the fingerprinting feature F, i.e., sequence of encrypted resource sizes. As discussed in Section 4, the values of this feature vary with different browsing contexts. We now discuss how the COLLECT_SAMPLES procedure handles these different scenarios.

1) **Factors causing no variation in feature values.** As discussed in Section 4, the values of encrypted resource sizes remain invariant to changes in factors such as network conditions. Snoopy does not require traffic samples to account for such factors that do not affect the feature values.
2) **Factors causing predictable variation in feature values.** As discussed in Section 4, the variations in encrypted resource sizes due to factors such as operating system, browser, and number of parallel tabs used are deterministic in nature. Differences in feature values due to these factors can be estimated by Snoopy from the domain knowledge of network and browser protocols incorporated in its design. Therefore, no extra traffic samples need to be collected for accounting for such browsing contexts. The COLLECT_SAMPLES procedure collects traffic samples for any one browsing context (for example, Firefox browser). Snoopy can estimate the feature values for the other browsing contexts (for example, Google Chrome browser) using static analysis techniques described in Section 5.2.
3) **Factors causing website-specific variation in feature values.** Snoopy requires traffic samples to estimate feature values due to variations in factors that depend on website design – the resources that can be cached and the information that the cookie carries. To handle this, the COLLECT_SAMPLES procedure collects traffic samples from each webpage once with caching on and cookies allowed and once with caching off and cookies prohibited.

To collect the traffic samples for estimating the feature values, the COLLECT_SAMPLES procedure spawns multiple dummy clients to access the webpages, one at a time, in separate sessions. During each webpage access, the encrypted traffic of each client is captured by a network monitoring tool from the beginning till the end of the browsing session. Subsequently, Snoopy processes every traffic trace and splits them into multiple sub-traces, corresponding to each resource r_i. Note that the packets carrying the same resource can be easily identified from an encrypted trace, since they have the same TCP acknowledgement number. The actual resource that is carried by the packets in a sub-trace are found by decrypting the sub-trace with the Transport Layer Security (TLS) keys used by the dummy clients. The sub-traces and the resources they correspond to are stored in the dataset named $TSamples$, where each entry is of

TABLE 3
Characteristics of Profiled Websites

Sl No	Website	HTTP version	TLS version	No of pages	Type of webpages
1	RS	1.1	1.2	20	plain HTML
2	SBC	2.0	1.3	27	HTML scripts
3	IC_1	1.1	1.2	95	aspx
4	B_1	1.1	1.2	444	aspx
5	B_2	1.1	1.2	458	HTML scripts
6	B_3	1.1	1.2	549	aspx
7	B_4	1.1	1.2	965	HTML + JS
8	IC_2	1.1	1.2	849	HTML + JS
9	B_5	1.1	1.2	1964	plain HTML
10	PS	2.0	1.2	40,323	HTML + JS

RS - Retail Store website, SBC - Service Based Company website, IC - Insurance Company website, B - Bank website, PS - Political Survey website.

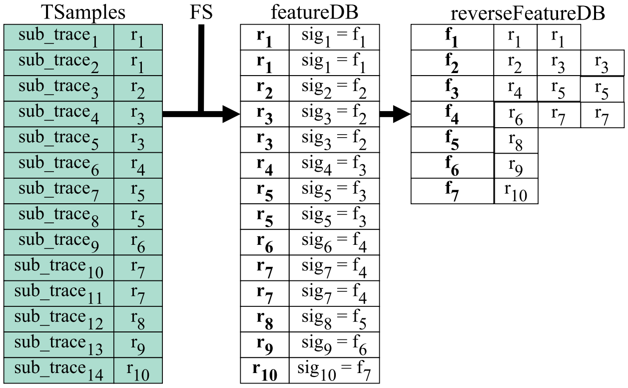

Fig. 4. Constructing featureDB and reverseFeatureDB from TSamples and FS.

the form {sub-trace, r}. Once the $TSamples$ dataset is built, the feature values of the resources can be extracted for building the resource signatures.

5.1.3 Constructing Feature and Resource Databases

In Step ⑤ of Algorithm 1, the BUILD_SIGNATURES procedure extracts the F value (encrypted size) of each resource from its corresponding sub-trace, which essentially captures the effect of the encryption function EF. For instance, if a sub-trace contains two packets of size 50 bytes each, the encrypted size of the corresponding resource would be 100 bytes. For every extracted sub-trace in $TSamples$, an entry of the form $< r_i, sig_i >$ is made in the signature database $featureDB$, where sig_i is the feature value of resource r_i.

During prediction, an inverse mapping (i.e., EF^{-1}) from an observed feature value (f_i) to the corresponding resource (r_i) will be required. However, constructing the inverse mapping EF^{-1} is not straightforward.

Challenges in Constructing EF^{-1}. Constructing EF^{-1} is challenging because it is not possible to perform a one-to-one mapping between encrypted resource sizes and the corresponding resources. This is because, as we discussed in Section 4, the sizes of encrypted resources vary due to variations in user browsing contexts. Therefore, a single resource might have different sizes in different browsing contexts. Likewise, a given size value might correspond to multiple resources in a given browsing context. Moreover, due to the finite query model, Snoopy cannot collect traffic samples (data points) for all possible browsing contexts.

The CONSTRUCT_DICTIONARY procedure builds a reverse database $reverseFeatureDB$ out of $featureDB$ by taking into account the many-to-many mapping between encrypted resource sizes and resources. Each entry in $reverseFeatureDB$ is of the form $< f_i, L_i >$, where f_i corresponds to a unique FS value and L_i is a list of resources that have this value. Note that L_i may include (1) multiple resources that have the same signature and (2) multiple instances of the same resource (each corresponding to a dummy client access). Fig. 4 shows an instance of featureDB and reverseFeatureDB constructed from the $TSamples$ collected corresponding to a website.

During prediction, to narrow down on the possible resources corresponding to a sub-trace, Snoopy uses the meta information (i.e., user-agent string, cache-ability, and cookies) to estimate the possible encrypted resource sizes in a given browsing context. Although the estimations for cache-ability and user-agent string can be determined easily (refer Section 5.1.2), for cookies, we need to collect additional traffic samples. The COMPUTE_COOKIE_VAR procedure (Step ⑦ of Algorithm 1) processes these traffic samples to extract and store the meta information pertaining to cookies as follows:

(1) Characterizing Tracking Cookies. Tracking cookie is an integral part of the source code of a webpage that records the sequence of webpages previously visited by a user. To account for the impact of tracking cookies on the size of encrypted resource r_i, we model the parameter $c_t(r_i)$. Note that there is no tracking cookie during profiling time, since each webpage is accessed in an individual session. The possible variation in feature values that arises due to embedded tracking cookies in a user trace is calculated by parsing the source code of the web resource (if it is in text format). Recently developed tools such as CookieCheck [45] can also be used for automated detection of tracking cookies in web resources. Note that only resources that are in text format carry tracking cookies. For each web resource that carries a tracking cookie, the COMPUTE_COOKIE_VAR procedure first identifies the set of URLs from which a user can navigate to the resource. Subsequently, for each of these URLs, it first computes the size of the resource *with* the tracking cookie when it is not encrypted. To compute the corresponding resource size when encrypted, we use linear interpolation of known plain-text and encrypted-text pairs. Finally, for each resource r_i, the estimated variation is stored as

$$c_t(r_i) = \{ < URL_1, \quad c_t(r_i, 1) >, < URL_2, c_t(r_i, 2) >, \\ \ldots, < URL_x, c_t(r_i, x) > \}. \tag{1}$$

Here, $c_t(r_i, j) (1 \leqslant j \leqslant x)$ denotes the increase in encrypted size of resource r_i due to an embedded tracking cookie carrying the string URL_j.

(2) Characterizing Session Cookies. Session cookies are transmitted when a page is accessed for the first time in a session. During profiling, the resources will always have the session cookie. The parameter $c_s(r_i)$ accounts for the discrepancy in the encrypted size of resource r_i due to the possible absence of session cookie in the real user's trace. This happens when the resource is not the first resource to be accessed by the real user. The value of $c_s(r_i)$ can be estimated by considering all the information carried in the header field (e.g., user ID, user agent, max-age of the cookie, its expiry date, etc.) and their possible values. Most of these information are of fixed length, except for the user agent field that contains the browser and OS name. Much like tracking cookies, the COMPUTE_COOKIE_VAR stores this information for each resource r_i as

$$c_s(r_i) = \{ < bo_1, \quad c_s(r_i, 1) >, < bo_2, c_s(r_i, 2) >, \\ \ldots, < bo_x, c_s(r_i, x) > \}. \tag{2}$$

Here $c_s(r_i, j) (1 \leqslant j \leqslant x)$ denotes the increase in encrypted size of resource r_i due to the session cookie carrying browser-OS identifier string bo_j.

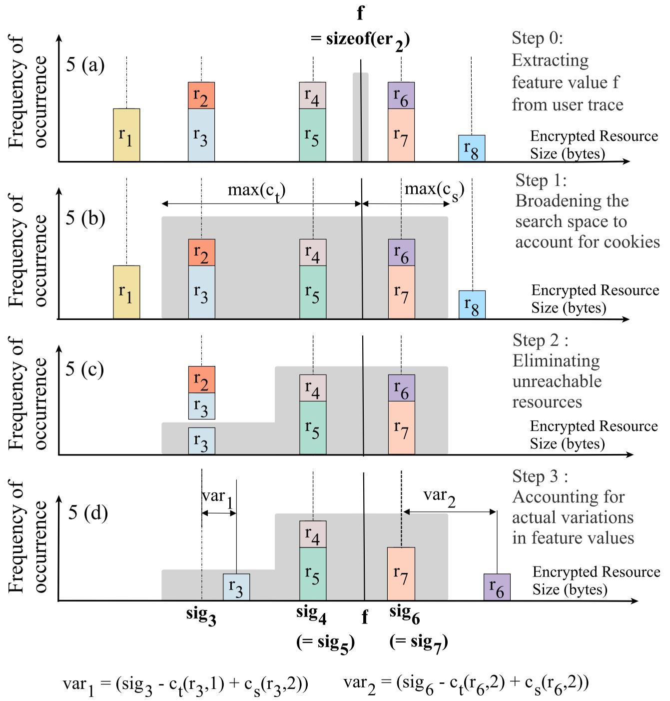

Fig. 5. Steps involved in selecting resources that might correspond to a given feature value.

The cookie induced variations thus estimated, are stored in a lookup-table $cookie\_var$, where each entry is of the form $< r_i, \{c_t(r_i), c_s(r_i)\} >$. For resources that do not contain tracking cookies, $c_t(r_i)$ contains $NULL$ value.

5.2 Webpage Prediction Module

The *Webpage Prediction Module* of Snoopy takes an encrypted traffic trace T as input, as shown in Step 3 of Fig. 2, and predicts the webpages that are accessed in T using the information stored in the Snoopy database. Snoopy first predicts the web resources accessed and subsequently uses them for predicting the webpages accessed in T.

5.2.1 *Predicting Web Resource Sequence*

For predicting the resources in the input trace T, Snoopy first splits T into sub-traces, each corresponding to an encrypted resource. From each sub-trace, the feature values are extracted and stored, in the order they appear in T, into an array $FValues$, where each element contains the feature value (encrypted resource size) of an encrypted resource er_i.

As a next step, Snoopy identifies the actual web resource that may correspond to each feature value in the array $FValues$, serially. It does so by looking-up the dictionary $reverseFeatureDB$. Apart from the fact that multiple resources may have the same feature value, the look-up procedure is also not straightforward due to variations in feature values induced by tracking and session cookies.

To start with, we organize the information available (for the website shown in Fig. 3) in the $reverseFeatureDB$ (refer to Fig. 4) as a stacked bar plot shown in Fig. 5a. The X-axis represents encrypted resource size and the Y-axis represents the frequency of occurrence of the resources with a particular size in the target website. For e.g., in Fig. 5a, r_7 and r_6 are stacked together and have the same size but r_7 (embedded in two webpages) is twice as likely to occur as compared to r_6 (embedded in only one webpage), as indicated by the height of the stacked bar. This is because, r_7 is embedded in two webpages, whereas r_6 is embedded in only one webpage. We begin the look-up process by considering a shaded region (around $x = f$ as shown in Fig. 5a) that represents the area in which the actual resource accessed by the user in trace T could be present. Subsequently, we refine our search by expanding and narrowing down the shaded region in each step.

We now illustrate the look-up with a concrete example.

Example. Consider two encrypted resources er_1 and er_2 extracted from T such that er_1 is accessed by the user before er_2. The feature values extracted from these two resources are stored in the first two elements of the array $FValues$ and processed sequentially. We assume without loss of generality that Snoopy has predicted that the resource, say, r_4, corresponds to the first encrypted resource er_1. We now describe how Snoopy processes the feature value of the second resource er_2. Let us assume the second value stored in the array $FValues$ is f. Algorithm 2 shows the different steps of the process.

Step ①: Broadening the search space –The goal of this step is to broaden the space in which we search for the resource corresponding to f (shown in Fig. 5a), since tracking and session cookies may have changed the resource sizes. If f is inclusive of the tracking cookies, it needs to be decremented, and if f does not contain session cookies, it needs to be incremented before performing a $reverseFeatureDB$ look-up. However, the presence or absence of these cookies cannot be determined by Snoopy at this stage, since it does not yet have the knowledge of the webpage from which the user navigated to this encrypted resource. Moreover, the identity of the resource is also not known. To account for these unknown parameters, we use the maximum possible values for tracking cookie and session cookie induced variations ($max(c_t)$ and $max(c_s)$ respectively) in encrypted resource size, retrieved from $cookie\_var$. Therefore, Snoopy performs a look-up of all values in the range $[(f - max(c_t)), (f + max(c_s))]$ and adds all the retrieved resources to the *multi*set $relevant\_resources$. Multiple instances of a resource are added to $relevant\_resources$ if the resource is associated with multiple webpages.

Step 1 of Fig. 5 shows that although f is closest to the sizes of r_4, r_5, r_6, and r_7, it could also possibly belong to r_2 or r_3 due to the aforementioned variations. These resources form the multiset $relevant\_resources = \{r_2, r_3, r_3, r_4, r_5, r_5, r_6, r_7, r_7\}$, and is indicated by the shaded region in Step 1 of Fig. 5.

Step ②: Reachability checking – In the real-world, a user starts browsing from a particular webpage and navigates using links embedded in the initial and subsequent webpages. Leveraging this, in this step Snoopy eliminates the resources that are not navigable from previously accessed webpages. For checking reachability, the CHECK_REACHABILITY method leverages the website structure $\mathbb{G}$ and the resource map $\mathbb{RM}$, and unlike prior works [14], [16], Snoopy also considers multi-tab browsing. Snoopy eliminates those resources that are not accessible from *any* of the previously accessed webpages from any of the open tabs.

In the example, er_1 ($= r_4$) is the previously accessed resource, which is uniquely associated to the webpage w_3

(Refer to website structure in Fig. 3). Through reachability check, we can infer that the resource er_2 can come from either of w_3, w_4, or w_5, and not from w_1 or w_2. If there is any resource in the $relevant\_resources$ set that is associated with w_1 (we do not have r_1 in this example) or w_2 (i.e., r_2 and r_3), its frequency should be reduced accordingly, i.e., the shaded region shrinks in size. Therefore, the set of reachable resources now contains $\{r_3, r_4, r_5, r_5, r_6, r_7, r_7\}$, denoted by $reachable\_resources$, indicated by the shaded region in Step 2 of Fig. 5.

Algorithm 2. PREDICT_RESOURCE

input: f, $\mathbb{G}$, $\mathbb{RM}$, $cookie\_var$, $reverseFeatureDB$, $predicted\_resources$
output: $predicted\_resources$

```
/ * input predicted_resources = ∅ when f corresponds to the first resource in T * /
/ * find set of all possible resources whose encrypted size might be equal to f after addition of tracking cookie or removal of session cookie * /
```
1: $relevant\_resources = \{reverseFeatureDB(x), \forall x \mid (f - max(c_t)) \leq x \leq (f + max(c_s))\}$;
```
/ * filter out unreachable resources * /
```
2: $reachable\_resources$ = CHECK_REACHABILITY ($relevant\_resources$, $predicted\_resources$, $\mathbb{G}$, $\mathbb{RM}$);
```
/ * estimate actual cookie-induced variation for each resource based on their reachability from different URLs. Update the list of resources to be considered accordingly. * /
```
3: $reachable\_resources$ = UPDATE_RESOURCE_LIST ($reachable\_resources$, $cookie\_var$);
```
/ * identify the most likely resource * /
```
4: $predicted\_resources$ = $predicted\_resources$.append(IDENTIFY_RESOURCE ($reachable\_resources$, f))

Step ③ : Accounting for cookie-induced variations in reachable resources – The goal of this step is to estimate the actual encrypted feature value for each resource r_i in the aforementioned step, based on information from $reverseFeatureDB$ and $cookie\_var$. For this, the UPDATE_RESOURCE_LIST method computes $(sig_i - c_t(r_i, x) + c_s(r_i, y))$ for each resource r_i in $reachable\_resources$, and eliminates all resources whose updated feature value falls outside the range under consideration. Snoopy estimates (1) $c_t(r_i, x)$ – which is the length of the URL contained in its tracking cookie. When the resource is associated with multiple URLs, some can be eliminated through reachability checking; and, **(2)** $c_s(r_i, y))$ – the browser and OS information contained in the session cookie in the Application layer header can be used. This information (i.e., bo_y from Section 5.1), can be estimated by using existing browser fingerprinting techniques [46] and OS fingerprinting techniques [47].

This step further narrows down the search space. To illustrate this step, we shift all the reachable resources in Fig. 5c, as shown in Fig. 5d. Note that the shaded region does not change in this step. The resource r_6 moves out of the shaded region whereas r_4, r_5 and r_7 remain unchanged since they have no cookie induced variation. However, note that r_3 is still under consideration, since it lies within the broadened search space even when its cookies are accounted for.

Step ④ : Assigning Weightage to Resources –In this step, Snoopy identifies the resource that is most likely to correspond to $er_2 = f$. For this, Snoopy assigns a weightage to each resource in $reachable\_resources$ from the previous step. The IDENTIFY_RESOURCE method measures the proximity of each of these resources to f, given that their cookie induced variations are accounted. It does so by taking a weighted average of their distance from f. Finally, Snoopy considers the resource with the highest weightage to be the one corresponding to er_2. In this example, er_2 is predicted as r_7. The output of this step is the sequence of resources in T as predicted by Snoopy, denoted as $predicted\_resources$.

5.2.2 Predicting Webpages Accessed by the User

Snoopy uses the longest common sub-sequence matching algorithm for predicting accessed webpages from $predicted\_resources$. Snoopy starts with the first resource (say, r_{p1}), and from the sequence set $\mathbb{S}$, it identifies all the webpages where r_{p1} is the first resource in the resource download sequence. The same process is followed recursively for the subsequent resources in $predicted\_resources$ until the webpage with the longest sub-sequence match is found. When Snoopy identifies the webpage, it adds the webpage to a set denoted as $predicted\_webpages$. Note that $predicted\_resources$ may not be empty yet since it could have the resources corresponding to multiple webpages. In such a case, the process is repeated.

Webpage identification becomes more challenging when caching is enabled in the browser. When caching is enabled, all the resources in the sequences in $\mathbb{S}$ for a webpage would not get downloaded, and thereby forming a gap in the sequence $predicted\_resources$. To handle such cases, during matching, we allow the algorithm to skip resources in $\mathbb{S}$ of a webpage as long as it is a cache-able resource and it has been downloaded in the past.

6 Implementation and Evaluation

We first describe our implementation and define the metrics used for evaluating Snoopy. Thereafter, we describe the experimental setup, which includes the websites used for evaluation, the user browsing scenarios considered for evaluation, and the traffic sample collection methodology. Subsequently, we evaluate Snoopy by answering a set of research questions.

6.1 Implementation

We implemented the functional modules of Snoopy as Python libraries. We also compared Snoopy with relevant state-of-the-art works on webpage fingerprinting such as ML_PS [14], ML_IPS [14], ML_OPS [14], ML_BoG [16], ML_LL [36], ML_KFP [18], ML_Wfin [21], and ML_CUMUL [19]. For this, we implemented these techniques to the best of our abilities with the help of open-source Python libraries [48]. While most of these ML-based techniques [14], [16], [36] were solely designed for the purpose of webpage fingerprinting over HTTPS, few others were not designed with the exact same goal. Techniques such as k-FP [18] and Wfin [21] were originally used for both website fingerprinting (over Tor) and webpage fingerprinting (over HTTPS). For our purpose, we only consider the following scenario that was presented in the k-FP [18] and Wfin [21] papers: *webpage* fingerprinting over HTTPS. For instance, in this scenario, k-FP uses features such as packet size, packet direction, and packet ordering. We used the same features and classification algorithms in our implementation of k-FP. Another technique, namely CUMUL [19], attempted to perform both website and webpage fingerprinting over Tor (not HTTPS). For our purpose, we were only interested in the *webpage fingerprinting* part of the paper. The paper proposed the use of (i) a novel ML-based technique as well as (ii) the use of a novel set of features (Tor and TLS features). The paper concedes that CUMUL was unsuccessful in performing webpage fingerprinting over Tor. This was mainly attributed to Tor's obfuscation of resource sizes, which rendered the TLS features useless. We use CUMUL as benchmark in our experiments but on *HTTPS traffic* (i.e., a much weaker defense than Tor). We expected CUMUL to perform better against this weaker defense due to the absence of Tor's obfuscation of resource sizes. In our implementation, we used the same ML-based technique and the TLS features proposed in CUMUL but used them on HTTPS dataset.

Encrypted network traffic traces used for webpage fingerprinting and testing was captured using TShark(v2.6.6) packet capture tool, which was running on the network gateway, although the same could be done by the attacker at any intermediate network router in the real world.

6.2 Metrics

We define the metrics that we use for evaluating Snoopy and existing state-of-the-art webpage identification techniques as follows:

Generalization factors (GF): The set of factors related to user's browsing behavior that the webpage identification technique aims to generalize across. **GF** $\subseteq \{I, BC, T, O, B, N\}$, where I denotes user interest and thereby the set of webpages in the website that the attacker needs to fingerprint, BC denotes the set of browser configurations (cache and cookie settings) considered by the attacker, T denotes the number of tabs used by a user in a browsing session, O indicates the set of Operating Systems that the attacker generalizes across, B denotes the set of Browsers, and N denotes network conditions.

Number of Queries ($\mathbf{N_q}$): The maximum number of website queries (accesses) that an attacker can perform on the targeted website for collecting traffic samples for training (webpage profiling).

Fingerprinting Accuracy (FA): The percentage of webpages accessed by a user (of the target website) in a browsing session that are correctly identified by a webpage identification technique.

6.3 Experimental setup

Websites Used. We evaluate Snoopy and related works [14], [16], [18], [19], [21], [36] on 20 websites that include some of the Fortune 100 companies, financial organizations, and service-based companies from different parts of the world. The scale of our experiments is at par with existing works on webpage identification [13], [16], [18], [20], [22], [31], [32], [34], [49]. As mentioned earlier, this paper deals with *webpage identification* performed *on a website* of interest to the adversary, in contrast to website identification that concerns thousands of websites. Due to space constraints, we present the results for ten of these websites, listed in Table 3, that are representative of the entire set. The names of these websites are anonymized to protect the websites from being targets of this attack.

User Browsing Scenarios. We consider different combinations of (1) Operating Systems viz.; Ubuntu 18.04, and Microsoft Windows 7, (2) Browsers viz.; Mozilla Firefox 63.0.3, and Google Chrome 67.0, (3) cache settings viz.; ON and OFF, (4) cookie preferences viz.; Allowed and Prohibited, and (5) network conditions. Users can start browsing from any webpage in the website that they would be interested in, and browse the pages in any order. Further, we also allow users to freely browse up to 15 different webpages in each browsing session, either sequentially or by using multiple parallel tabs, with no restriction on the transition time from one webpage to another. Most existing works do not consider cases where the users browse more than one page in a session and even if they do, they restrict it to a small number(for e.g., 4 in [20] and 2 in [17]). These numbers are much lower than the average number of parallel browser tabs used by website visitors, as pointed out in a study[3] conducted by Mozilla [50].

Traffic Sample Collection. We developed a Python bot for collecting encrypted traffic samples for evaluating Snoopy as well as the existing works. Our traffic collection bot used Selenium for simulating behavior of real website users while collecting test traffic samples. For creating webpage fingerprints, our bot accessed the webpages as many times as required by the different webpage fingerprinting techniques, within the limits of the finite query model. Depending on the requirements of our experiments, the bot performs either sequential or single-page traffic sample collection. We introduced a delay of 1 minute between subsequent browsing sessions. We observed that repeated website accesses from the same network to the same website needed to be separated by this time in order for it to not be flagged and blocked by the website. For evaluations that required sequential browsing traffic samples, the browsing sequence length was varied from 3 to 15 webpages per session. For evaluations that required traffic samples collected over various network conditions, the traffic sample collection was conducted over several months and from different geographical locations to ensure variations in network conditions. In addition to the webpage(s) accessed in a browsing session (which is needed for computing the accuracy), our bot also recorded the sequence of web objects downloaded in each browsing session, which we use for additional evaluation.

6.4 Results

In this section, we first evaluate the suitability of Snoopy in the context of practical mass-surveillance. In the prior sections, we have discussed about the two key requirements for conducting practical mass-surveillance – generalization

3. Mozilla has removed the dataset compiled for the Test Pilot study conducted in 2010 from public domains. But we can speculate that the trend of tabbed webpage browsing has only gone up in the last decade.

TABLE 4
Webpage Prediction Accuracy of Snoopy

Website	No. of webpages	Webpage identification accuracy		
		Accurately identified (%)	Not identified (%)	Wrongly identified (%)
IC_1	95	93	7	0
IC_2	849	89	11	0
B_1	444	99	1	0
B_2	458	99	1	0
B_3	549	97	0	3
B_4	965	88	11	1
B_5	1964	90	3	7
SBC	27	81	7	12
RS	20	75	21	4
PS	40323	83	17	0

and compliance with a finite query model. Therefore, we primarily evaluate Snoopy on these two parameters, through experiments that intend to answer the following questions.

Q1) How effective is Snoopy in different browsing contexts? (Refer to Section 6.4.1)
Q2) How is the effectiveness of Snoopy and related works affected with changes in the value of N_q (number of queries allowed)? (Refer to Section 6.4.2)
Q3) How well does Snoopy generalize across user interests? (Refer to Section 6.4.3)
Q4) How well does Snoopy generalize across different configurations of a browser? (Refer to Section 6.4.4)
Q5) How well does Snoopy generalize in terms of number of tabs used by the end user while browsing? (Refer to Section 6.4.5)
Q6) How well does Snoopy generalize in terms of variations in Operating Systems? (Refer to Section 6.4.6)
Q7) How well does Snoopy generalize in terms of variations in Browsers? (Refer to Section 6.4.7)
Q8) How well does Snoopy generalize across various network conditions? (Refer to Section 6.4.8)

6.4.1 Average Prediction Accuracy of Snoopy Across Diverse Browsing Scenarios

We now show the prediction accuracy (FA) of Snoopy when it builds webpage fingerprints (trains) for any one browsing scenario and uses this information to predict user activities across diverse browsing scenarios. For this experiment, we considered all the 10 websites. For each website, we built a training dataset where we consider $GF = \{I, BC, T, O, B, N\}$ such that, I = {all webpages in the website}, $BC = \{\{$Caching ON, Cookies Allowed$\}, \{$Caching OFF, Cookies Prohibited$\}\}$, $T = 1$, $O = \{Ubuntu\}$, $B = \{Firefox\}$. N was kept constant by collecting all traffic samples from the same system within a short span of time. We collect 10 such traffic samples for each webpage.

For each website, we also build a test dataset, where we consider I = {all webpages in the website}, vary BC across the values $\{\{$Caching ON, Cookies Allowed $\}, \{$Caching ON, Cookies Prohibited$\}, \{$Caching OFF, Cookies Allowed$\}, \{$ Caching OFF, Cookies Prohibited$\}\}$, vary T from 1 to 15, vary O across the values $\{Windows, Ubuntu\}$, and vary B across the values $\{Chrome, Firefox\}$. The network conditions N were varied by collecting traffic samples over several months from different geographical locations (refer to the test dataset described in Section 6.3).

Table 4 shows the webpage prediction accuracy (FA) of Snoopy on the 10 websites. The FA value is more than 90% for most of the websites, indicating the ability of Snoopy to generalize across different browsing contexts. In the rest of this section, we show how the different factors in GF individually influence the FA value of Snoopy, when subjected to constraints on the value of N_q, the maximum number of queries allowed to a website.

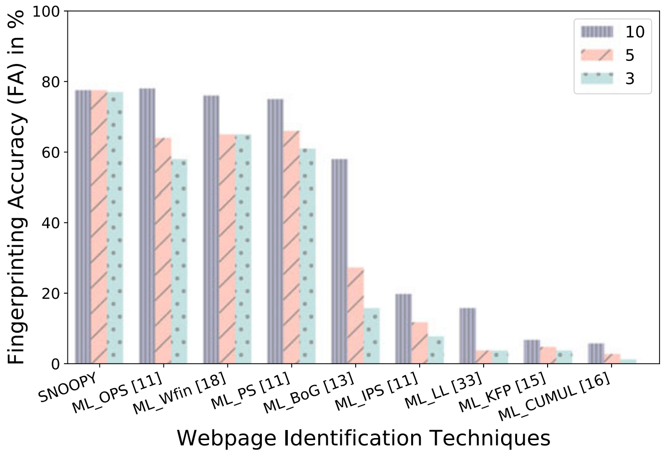

Fig. 6. Importance of training set size N_q on fingerprinting accuracy FA (in %) of existing ML models versus Snoopy (for 200 web pages).

6.4.2 Compliance With a finite Query Model: Snoopy versus ML-Based Solutions

We now evaluate Snoopy on its ability to comply with a finite query model with respect to existing works. For this experiment, we considered the website B_4 and generalization factors $GF = \{I, BC, T, O, B, N\}$, where I is a set of 200 random webpages of the website B_4, $BC = \{\{$Caching OFF, Cookies Allowed$\}\}$, $T = 1$, $O = \{Ubuntu\}$, $B = \{Firefox\}$. N was constant since the traffic samples were collected within a short span of time. Our test dataset comprised traffic traces from the website B_4 collected using the above configuration. We built three different training datasets with traffic samples collected using the same configuration, but varying the number of samples collected per webpage. We set N_q, the maximum number of queries allowed, as $w \times s$, where w is the number of webpages to be fingerprinted and s is the number of traffic samples collected per webpage. Keeping $w = 200$ constant, we vary the value of s as $s = 10, 5,$ and 3.

Fig. 6 plots the fingerprinting accuracy (FA) for Snoopy and existing works for different values of s. From the figure, we can observe that (1) When $s = 10$, the prediction accuracy of Snoopy (77.75%) is at par with the existing ML-based techniques, and; (2) When $s = 5$ or $s = 3$, we observed a drop in FA of ML techniques (for e.g., from 78% to 58% in case of ML_OPS [14]) . On the other hand, we can see that FA for Snoopy remains unchanged even after lowering the

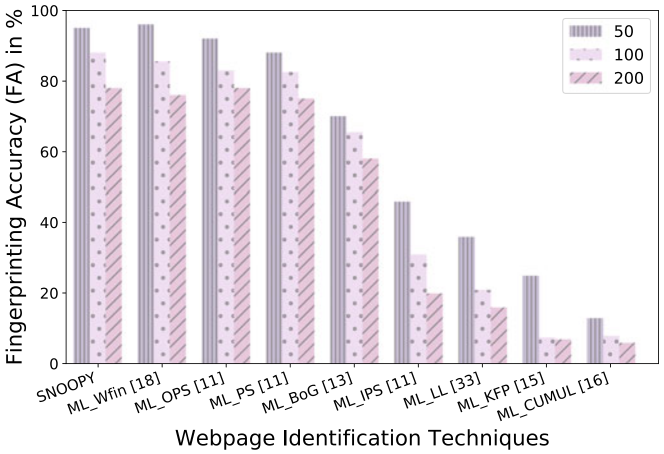

Fig. 7. Fingerprinting accuracy FA of Snoopy versus existing techniques for different number of webpages of interest.

value of s. Since we have seen that for the three different s values considered in this experiment, the ML models have the highest FA value when $s = 10$, we use $s = 10$ to constrain the value of N_q for the finite query model assumed in the rest of our experiments (shown in Sections 6.4.3, 6.4.4, 6.4.5, 6.4.6, 6.4.7, and 6.4.8).

6.4.3 Generalization Across User Interests

We now show how the fingerprinting accuracy (FA) of Snoopy and existing works change with the number of webpages of interest (I). For this experiment, we consider the website B_4 and generalization factor $GF = \{I, BC, T, O, B, N\}$, where we set $BC = \{\{\text{Caching OFF, Cookies Allowed}\}\}$, $T = 1$, $O = \{Ubuntu\}$, $B = \{Firefox\}$. N was constant since the traffic samples were collected within a short span of time. We varied I as sets of $50, 100,$ and 200 webpages for building three different training datasets. For this experiment, $N_q = |I| \times 10$, which is the constraint imposed by our finite query model. For each of the training datasets, we built test datasets using encrypted traffic traces with the same constraints as the training dataset.

Fig. 7 shows the fingerprinting accuracy (FA) of Snoopy and existing works when the number of webpages of interest increases. From the figure we can observe that (1) Despite using a simple feature and a simple approach, Snoopy gives the same prediction accuracy as some of the best-performing ML-based techniques (e.g., ML_Wfin [21]), and; (2) When $|I|$ was increased from 50 to 100 and 200, there was a fall in the value of FA for Snoopy as well as the existing ML-based techniques. Even then, Snoopy gave the same prediction accuracy as the best-performing ML-based techniques (for e.g., ML_Wfin [21] and ML_OPS [14]).

A detailed inspection revealed that the drop in the FA value of Snoopy when the size of I was gradually increased was due to the fact that many of the webpages in the larger sets had similar fingerprints. In Section 6.4.10, we will discuss the reason behind this in more details, and in Section 6.5 we will discuss possible ways to improve the fingerprinting accuracy (FA) of Snoopy.

TABLE 5
Accuracy (in %) of Snoopy versus ML Models When Tested on Data Points From a Different Browser Configuration (for 200 Webpages)

Classifier	Browser Cache Configuration	
	Training: OFF Testing: ON	Training: OFF Testing: OFF
SNOOPY	**72**	**78**
ML_PS [14]	61	75
ML_OPS [14]	60	78
ML_Wfin [21]	51	76
ML_BoG [16]	50	58
ML_LL [36]	14	16
ML_KFP [18]	4.5	7
ML_IPS [14]	3	20
ML_CUMUL [19]	2.5	6

6.4.4 Generalization Across Browser Configurations

Next, we evaluate Snoopy and existing works on their ability to generalize across different configurations of caching and cookie settings for a given browser (BC). For this experiment, we consider the website B_4 and $GF = \{I, BC, T, O, B, N\}$, such that I is a set of 200 random webpages of the website B_4, $T = 1$, $O = \{Ubuntu\}$, $B = \{Firefox\}$, and N was constant. While building our training dataset, we keep $BC = \{\text{Caching OFF, Cookies Allowed}\}$ constant, and collect 10 samples of browsing traffic from each of the 200 webpages of B_4 that we considered. For this experiment, $N_q = 200 \times 10$, which is the constraint imposed by the finite query model. In our test dataset, we vary BC across the values $\{\{\text{Caching OFF, Cookies Allowed}\}, \{\text{Caching ON, Cookies Allowed}\}\}$, with all other factors similar to the training dataset.

Table 5 shows a comparison of the fingerprinting accuracy (FA) of Snoopy and related works. For the straightforward case, where training and testing used the same BC configuration, Snoopy achieves an FA that is comparable ($FA \approx 75\% - 78\%$) to the best performing ML models as expected. However, when the BC configurations for testing and training were different, Snoopy outperforms even the best ML techniques. For example, Snoopy achieves an $FA = 72\%$ as compared to ML_PS [14] that achieves $FA = 61\%$.

6.4.5 Support for Prediction on Multi-Tab Browsing Traffic

We now evaluate Snoopy in the context of a real-world scenario where the users of the targeted website open the webpages simultaneously in multiple parallel browser tabs (T). For this experiment, we considered the website B_4 and $GF = \{I, BC, T, O, B, N\}$, such that I is a set of 200 random webpages of the website B_4, $BC = \{\text{Caching OFF, Cookies Allowed}\}$, $O = \{Ubuntu\}$, $B = \{Firefox\}$, and N was constant. While building our training dataset, we keep $T = 1$ constant, and collect 10 samples of browsing traffic from each of the 200 webpages of B_4 that we considered. For this experiment, $N_q = 200 \times 10$, which is the constraint imposed by the finite query model. In our test dataset, we

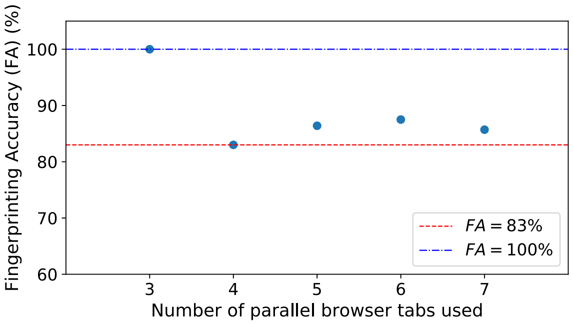

Fig. 8. Fingerprinting accuracy (FA) of Snoopy in multi-tab browsing scenario (for website B_4).

vary T from 3 to 7 with all other factors similar to the training dataset.

Fig. 8 shows average the fingerprinting accuracy (FA) of Snoopy for browsing sessions for the website B_4, involving different number of parallel browser tabs. We observe that even for 7 parallel browser tabs, the average fingerprinting accuracy of Snoopy is more than 85%.

The traffic samples collected for building the training dataset for Snoopy in this case correspond to single-tab single-webpage browsing sessions. This is in contrast to existing works [13], [14], [16], [20] that fingerprint sequences of webpage browsing traffic instead of single webpage accesses. While existing techniques have been shown to result in a high fingerprinting accuracy (FA), it magnifies the number of website accesses required (N_q) to a great extent. Table 6 shows a comparison of the number of webpage sequences of length $3-7$ for the 10 websites we selected for our experiments, with the number of individual webpages that Snoopy would access for building fingerprints. Existing works would require $N_q = l\times$ number of traffic samples for training their models, where l is the number of webpage sequences possible. On the other hand, Snoopy would require only $N_q = w\times$ number of traffic samples, where w ($\ll l$) is the number of webpages in the website. Note that the number of webpage sequences shown in Table 6 do not consider multiple occurrences of the same webpage in a sequence. Considering such cases would have further increased N_q for existing works. This might have caused the website to block the adversary from collecting traffic samples, and would have definitely escalated the time required for collecting traffic samples, and training the model. On the other hand, this would not impact the N_q value for Snoopy. Latest works [17] that are closest to Snoopy in terms of working principle have not been able to obtain a considerable FA value beyond 2 parallel tabs.

6.4.6 Generalization Across Various Operating Systems

We now evaluate Snoopy and existing works on their ability to generalize across different operating systems (O). In this experiment we consider the website IC_1 and $GF = \{I, BC, T, O, B, N\}$, such that I is the set of 95 webpages of the website IC_1, BC = {{Caching OFF, Cookies Allowed}}, $T = 1$, $B = \{Firefox\}$, and N was constant. We built two different training datasets, one where $O = \{Ubuntu\}$ was kept constant, and another where $O = \{Windows\}$ was kept

TABLE 6
Snoopy versus Existing Works: Number of Website Accesses Required for Fingerprinting in Multi-Tab Browsing Scenario

Website	Number of unique webpage sequences (Sequence Length: 3-7)l)	Minimum number of website accesses required by Snoopy$w$)
IC_1	586	95
IC_2	170,538	849
B_1	9,291,522	444
B_2	1199	458
B_3	13,617	549
B_4	7,000	965
B_5	103,116	1964
SBC	3,271	27
RS	1,159	20
PS	11	11

constant. For building each training dataset, we collected 10 samples of browsing traffic from each of the webpages in I. For this experiment, $N_q = 95 \times 10$, which is the constraint imposed by the finite query model. In our test dataset, we considered $O = \{Ubuntu\}$, with all other factors similar to the training datasets.

Table 7 shows a comparison of the fingerprinting accuracy (FA) of Snoopy and related works. When training and testing were performed using traffic samples from the same OS, Snoopy achieves an FA that is comparable ($FA \approx 94\% - 98\%$) to the best performing ML model ML_Wfin [21], as expected. However, when the OS used for testing and training were different, Snoopy outperforms even the best ML techniques by a huge margin. For example, Snoopy achieves an $FA = 97.8\%$ as compared to ML_Wfin [21] that achieves $FA = 54.8\%$.

6.4.7 Generalization Across Various Browsers

We now evaluate Snoopy and existing works on their ability to generalize across different browsers (B). In this experiment we consider the website IC_1 and $GF = \{I, BC, T, O, B, N\}$, such that I is the set of 95 webpages of the website IC_1, BC = {{Caching OFF, Cookies Allowed}}, $T = 1$, $O = \{Ubuntu\}$, and N was constant. We built two different training datasets, one where $B = \{Google\ Chrome\}$ was kept constant, and another where $B = \{Firefox\}$ was kept constant. For building each training dataset, we collected 10 samples of browsing traffic from each of the webpages in I. For this experiment, $N_q = 95 \times 10$, which is the constraint imposed by the finite query model. In our test dataset, we considered $B = \{Firefox\}$, with all other factors similar to the training datasets.

Table 8 shows a comparison of the fingerprinting accuracy (FA) of Snoopy and related works. When training and testing were performed using traffic samples from the same browser, Snoopy achieves an FA that is comparable ($FA \approx 94\% - 98\%$) to the best performing ML model ML_Wfin [21], as expected. However, when the OS used for testing and training were different, Snoopy outperforms even the best ML techniques by a huge margin. For example, Snoopy achieves an $FA = 85.9\%$ as compared to ML_Wfin [21] that achieves $FA = 13.8\%$.

TABLE 7
Accuracy (in %) of Snoopy versus ML Models When Tested on Data Points From a Different Operating System (for 95 Webpages)

Classifier	Operating System	
	Training: Windows Testing: Ubuntu	Training: Ubuntu Testing: Ubuntu
SNOOPY	**97.8**	**97.9**
ML_Wfin [21]	54.8	93.7
ML_CUMUL [19]	21.5	78
ML_KFP [18]	15.0	65.2
ML_BoG [16]	2.1	77.9
ML_PS [14]	1.1	77.9
ML_IPS [14]	1.1	62.1
ML_OPS [14]	1.1	55.8
ML_LL [36]	1	40.0

TABLE 8
Accuracy (in %) of Snoopy versus ML Models When Tested on Data Points From a Different Browser (for 95 Webpages)

Classifier	Browser	
	Training: Chrome Testing: Firefox	Training: Firefox Testing: Firefox
SNOOPY	**85.9**	**97.9**
ML_Wfin [21]	13.8	93.7
ML_PS [14]	8.0	77.9
ML_OPS [14]	8.0	55.8
ML_IPS [14]	5.7	62.1
ML_LL [36]	5.7	40.0
ML_CUMUL [19]	3.4	78
ML_BoG [16]	2.3	77.9
ML_KFP [18]	2.3	65.2

6.4.8 Generalization Across Network Conditions

We did not observe any changes in the fingerprinting accuracy of Snoopy due to small-scale natural network fluctuations. The results shown in Table 4 reflects the prediction accuracy of Snoopy across diverse network conditions. The traffic samples that were used for profiling were collected over a steady network connection within a short span of time. On the contrary, the test traffic traces were collected over a month from different geographical regions to ensure sufficient diversity in network conditions.

To further test the limits of Snoopy, we introduced artificial perturbations in the compromised network device over an hour by randomly adding delays of 50ms to 80ms and throttling the bandwidth by 20%. In this case, we encountered significant packet drops. This resulted in incomplete download of web resources on several instances, and consequently Snoopy could not identify the webpages correctly. The percentage of cases where the webpages could not be identified by Snoopy has been indicated in Table 4.

6.4.9 Resource-Level Prediction Accuracy of Snoopy

We now present more detailed evaluation results about the performance of Snoopy. This includes individual resource-level prediction accuracy of Snoopy and the correlation between resource-level prediction accuracy and webpage-level prediction accuracy. We also analyze the cases where Snoopy fails to accurately identify the webpages accessed, and propose potential solutions to effectively handle such scenarios.

Table 9 shows the resource prediction accuracy of Snoopy on the ten websites. While for most of the websites Snoopy had a prediction accuracy of more than 70%, in a few cases, we witnessed an accuracy of less than 50%. For websites with a very low number of resources (for instance, IC_1 and RS) Snoopy had a high prediction accuracy since the encrypted size of most of the resources were very distinct from each other. On the other hand, Snoopy had a low prediction accuracy for most websites with a high number of resources (for instance, B_2 and B_5). This is because, in practice, most of the web resources in such websites have similar sizes. However, the website IC_2 was an exception where Snoopy had a relatively high prediction accuracy despite a large number of

TABLE 9
Prediction Accuracy of Resources in Bot Traces

Website	No of resources	Resource identification accuracy		
		Accurately identified (%)	Unidentified resources	
			Incomplete download (%)	Conflict (%)
IC_1	187	94	5	1
IC_2	19,163	71	19	10
B_1	865	69	7	24
B_2	12,435	47	52	1
B_3	688	91	5	4
B_4	3472	83	1	16
B_5	3,998	68	26	6
SBC	277	81	7	12
RS	204	94	6	0
PS	55 (11#)	15 (76##)	-	0

#*The number indicates the resources that are of interest to the adversary.*
##*Accuracy computed with respect to the number of resources of interest.*

resources. This was because, most of the resources in this website had distinct sizes. Also, note that out of the 55 resources in the website PS, only 11 resources were of interest to the adversary, and their download sequence was sufficient to identify all the 40323 webpages uniquely. Details about this can be found in our recent work [44].

6.4.10 Relation Between Resource Prediction Accuracy and Webpage Prediction Accuracy

As seen from Tables 4 and 9, the relationship between web resource prediction and webpage prediction accuracy is not straightforward. We now discuss three different scenarios that we encountered during the evaluation of Snoopy.

Case 1: High resource prediction accuracy and high webpage prediction accuracy. In case of websites IC_1 and B_3, we observed a high correlation between web resource prediction accuracy ($91\% - 94\%$) and webpage prediction accuracy ($93\% - 97\%$). While such results are quite intuitive, such direct correlation was not observed in case of some other websites;

Case 2: Low resource prediction accuracy and high webpage prediction accuracy. In case of IC_2, B_1, B_2 and B_5, we see low values of resource prediction accuracy ($47\% - 75\%$) but relatively high values of webpage prediction accuracy ($89\% - 99\%$). The website B_2 exhibits an extreme case of this behavior with only 47% resource prediction accuracy but 99% webpage prediction accuracy. The reason for this is though the overall resource prediction accuracy was low, Snoopy was able to identify the critical resources that are unique to a webpage; and,

Case 3: High resource prediction accuracy and low webpage prediction accuracy. In case of the website RS, though the resource prediction accuracy was high (94%), the webpage prediction accuracy was relatively low (75%). This was because most of the web resources that Snoopy could predict correctly were non-critical resources that were shared by multiple webpages.

Failure Analysis – Our analysis reveals that the most common reasons due to which Snoopy could not perform well are (1) when the identified resources are associated with a lot of other webpages; (2) when the resources critical for identifying a webpage are already cached at the browser and do not get downloaded; and, (3) increase in number of possible webpage transitions in case of multi-tab browsing. For instance, Snoopy may correctly identify a web resource but may not be able to determine which tab it came from in case webpages open on multiple tabs at the same time share the resource. This increases the confusion and leads to incorrect predictions. To improve the webpage prediction accuracy in cases where Snoopy has a poor accuracy, we explore an ensemble of Snoopy and a ML-based technique in Section 6.5, that complies with the constraints of a finite query model.

6.5 Snoopy-ML Ensemble

Our experiments (Refer to Section 6.4) show that there are certain cases where the webpage prediction accuracy of Snoopy is approximately 80%. This is comparatively lower than most of the other websites where Snoopy could achieve a prediction accuracy of more than 90%. This motivated us to design a small experiment where we analyze the effectiveness of ML-based webpage identification techniques in classifying only those webpages that Snoopy fails to classify.

6.5.1 Experiment 1

For our experiment we consider the website B_4, and the scenario described in Section 6.4.3, where we study the generalization capability of Snoopy and ML-based techniques in terms of user interests. We had observed that for 200 webpages of website B_4 and 10 traffic samples per webpage, Snoopy had achieved a prediction accuracy of 78%. We first examine how the best-performing ML-based technique, ML_Wfin [21], performs when trained and tested on only those webpages that Snoopy failed to identify.

We first inspect the prediction results of Snoopy and identify 70 out of 200 webpages that Snoopy could not predict correctly. Next, we train the ML_Wfin model using 10 samples from each of these 70 webpages. During validation, the ML model assigns a probability to each of these 70 classes (webpages) for each validation point. For a given validation point (say, t_j), the webpage that gets assigned the highest probability (denoted as P_j) is the prediction output.

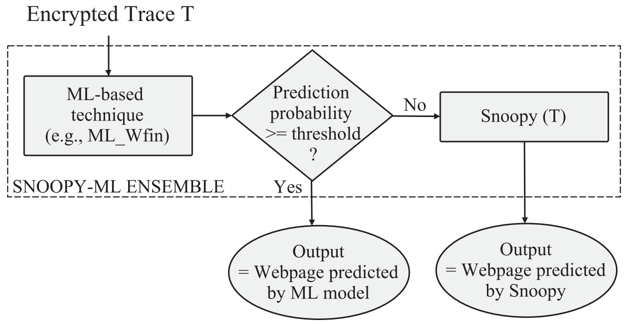

Fig. 9. Webpage prediction by Snoopy-ML ensemble.

For the 70 webpages, we observe a validation accuracy of $\approx 94\%$. The results of this experiment motivated us to explore if there is a way to combine Snoopy and an ML-based technique into an ensemble that outperforms each of the identification techniques individually; subject to the limitations on the number of queries.

6.5.2 Experiment 2

We build a very basic ensemble model with Snoopy and ML_Wfin [21] as sub-modules. We train (webpage profiling) Snoopy on 130 webpages of the website B_4, while we train ML_Wfin on the remaining 70 webpages of B_4. We use 10 traffic samples from each webpage for this training. Fig. 9 shows how this ensemble is used for predicting webpages accessed in a given encrypted trace T. The trace T is first passed to ML_Wfin for prediction. ML_Wfin predicts one out of the 70 webpages it has been trained with, with a probability P_{ML}. Based on our insights from Experiment 1 (refer to Section 6.5.1), we calculate a threshold probability $P_v = min(P_j), \forall_j$ (where, P_j is the maximum probability assigned to a class by ML_Wfin for validation point t_j) to determine if T belongs to one of the 70 webpages that ML_Wfin was trained with.

If $P_{ML} \geq (P_v - 10)$ (error margin = 10%), we consider the webpage predicted by ML_Wfin as the ensemble output. If $P_{ML} < (P_v - 10)$, the trace T is passed to Snoopy for prediction, and we consider the class predicted by Snoopy as the ensemble output. For the scenario we considered in Experiment 1, the observed value of P_v was 32.5%.

For testing this basic ensemble, we use a dataset comprising encrypted traffic traces from the website B_4, collected using the same browsing context as the training dataset. Our ensemble achieved a prediction accuracy of 97%, which is much higher than the accuracy achieved by Snoopy (78%) or ML_Wfin (76%) separately. Since in this experiment we considered the case where generalization is performed only with respect to user interests, we did not need additional training samples for ML_Wfin. However, note that we would have to collect more training samples for ML_Wfin if we considered other factors in $\mathbb{GF}$ as well. While this might not be the best possible ensemble, the fact that even such a basic ensemble achieved a high accuracy shows an interesting future research direction.

7 Countermeasures

Existing countermeasures [15], [51], [52], [53] perform traffic pattern obfuscation to prevent website/webpage fingerprinting attacks. Deployment of these countermeasures require the participation and co-operation of the website users. Website user participation was acceptable in these cases since these countermeasures were primarily designed for protecting the privacy of website users. However, the privacy leak presented in our work is more critical to organizations than the website users. In such a case, existing countermeasures cannot be deployed. Instead, we explore alternatives that can be deployed by the organizations, even without the participation of their website users.

Snoopy, although presented as a webpage fingerprinting technique, can also be seen as a tool that can be leveraged by organizations to design websites that are resilient to mass surveillance, before they are deployed in the public domain. Once a website has been designed, the web developer of the organization can set up a local test-bed to generate encrypted traffic traces corresponding to different browsing contexts that mimic real website users. The developer can then use Snoopy to determine if these traces are vulnerable. If so, the detailed prediction reports of Snoopy can help the developer understand which resources in the website leak information. Following are some of the ways in which the detailed report can be used for designing a countermeasure:

Altering the Web Resources. The web developer can alter the size of the vulnerable web resources (e.g., images, videos, texts) and/or change the sequence in which they get downloaded. Web resource sizes can be changed either by padding them with extra bytes or by re-rendering them from scratch. For changing the download order, the developer can either modify the HTML or, when HTTP/2 deployment is available, change the download priority of resources.

Restructuring the Website. Restructuring websites can sometimes help minimize the impact of surveillance on the privacy of the organization. For instance, a news website that features one news article per webpage can be restructured by grouping multiple articles in a webpage. Although Snoopy may still infer the webpage accessed, the information may not be useful in estimating the popularity of news articles on that webpage.

8 Conclusion

In this article, we proposed Snoopy, a webpage fingerprinting framework for performing mass surveillance while complying with a finite query model. Snoopy achieves this objective with $\approx 90\%$ accuracy, across various browsing contexts, while requiring only $3-10$ traffic samples per webpage. Furthermore, for cases where the standalone Snoopy framework was not sufficient, we presented a preliminary Snoopy-ML ensemble model that achieved $\approx 97\%$ accuracy. We believe that this paper will motivate researchers to design countermeasures against ETA-based mass-scale privacy attacks.

References

[1] Personal data of 533m Facebook users leaked for free on the Dark Web, 2021. Accessed: Mar. 22, 2022. [Online]. Available: https://www.teiss.co.uk/533m-facebook-users-data-leaked-dark-web/

[2] Cyber criminals leak personal data of 2.9 cr Indians on dark web for free, 2020. Accessed: Mar. 22, 2022. [Online]. Available: https://economictimes.indiatimes.com/tech/ites/cyber-criminals-leak-personal-data-of-2–9-cr-indians-on-dark-web-for-free/articleshow/75904331.cms

[3] Facebook–Cambridge Analytica data scandal, 2018. Accessed: Mar. 22, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Facebook–Cambridge_Analytica_data_scandal

[4] Y.-W. Seo and B.-T. Zhang, "Learning user's preferences by analyzing web-browsing behaviors," in *Proc. 4th Int. Conf. Auton. Agents*, 2000, pp. 381–387.

[5] S. E. Middleton, D. C. De Roure, and N. R. Shadbolt, "Capturing knowledge of user preferences: Ontologies in recommender systems," in *Proc. 1st Int. Conf. Knowl. Capture*, 2001, pp. 100–107.

[6] J. Hong, "The state of phishing attacks," *Commun. ACM*, vol. 55, no. 1, pp. 74–81, 2012.

[7] V. L. Le, I. Welch, X. Gao, and P. Komisarczuk, "Anatomy of drive-by download attack," in *Proc. 11th Australas. Inf. Secur. Conf.*, 2013, pp. 49–58.

[8] A. Hussain, J. Heidemann, and C. Papadopoulos, "A framework for classifying denial of service attacks," in *Proc. Conf. Appl. Technol. Architectures Protoc. Comput. Commun.*, 2003, pp. 99–110.

[9] S. Goel, J. M. Hofman, and M. I. Sirer, "Who does what on the web: A large-scale study of browsing behavior," in *Proc. Int. AAAI Conf. Web Soc. Media*, 2012, pp. 130–137.

[10] L. Olejnik, C. Castelluccia, and A. Janc, "Why Johnny can't browse in peace: On the uniqueness of web browsing history patterns," in *Proc. Workshop Hot Topics Privacy Enhancing Technol.*, 2012, pp. 48–63.

[11] H. Müller and A. Sedley, "HaTS: Large-scale in-product measurement of user attitudes & experiences with happiness tracking surveys," in *Proc. Future Des.*, 2014, pp. 308–315.

[12] S. Bird, I. Segall, and M. Lopatka, "Replication: Why we still can't browse in peace: On the uniqueness and reidentifiability of web browsing histories," in *Proc. 16th USENIX Conf. Usable Privacy Secur.*, 2020, pp. 489–503.

[13] 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

[14] P. Chapman and D. Evans, "Automated black-box detection of side-channel vulnerabilities in web applications," in *Proc. 18th ACM Conf. Comput. Commun. Secur.*, 2011, pp. 263–274.

[15] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: Website fingerprinting attacks and defenses," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 605–616.

[16] 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 *Proc. Int. Symp. Privacy Enhancing Technol. Symp.*, 2014, pp. 143–163.

[17] X. Gu, M. Yang, and J. Luo, "A novel website fingerprinting attack against multi-tab browsing behavior," in *Proc. IEEE 19th Int. Conf. Comput. Supported Cooperative Work Des.*, 2015, pp. 234–239.

[18] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *Proc. 25th USENIX Conf. Secur. Symp.*, 2016, pp. 1187–1203.

[19] A. Panchenko et al., "Website fingerprinting at internet scale," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2016.

[20] Y. Xu, T. Wang, Q. Li, Q. Gong, Y. Chen, and Y. Jiang, "A multi-tab website fingerprinting attack," in *Proc. 34th Annu. Comput. Secur. Appl. Conf.*, 2018, pp. 327–341.

[21] J. Yan and J. Kaur, "Feature selection for website fingerprinting," *Proc. Priv. Enhancing Technol.*, vol. 2018, no. 4, pp. 200–219, 2018.

[22] H. F. Alan and J. Kaur, "Client diversity factor in HTTPS webpage fingerprinting," in *Proc. 9th ACM Conf. Data Appl. Secur. Privacy*, 2019, pp. 279–290.

[23] V. Ghiëtte and C. Doerr, "Scaling website fingerprinting," in *Proc. IFIP Netw. Conf.*, 2020, pp. 199–207.

[24] M. Shen, Y. Liu, L. Zhu, X. Du, and J. Hu, "Fine-grained webpage fingerprinting using only packet length information of encrypted traffic," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 2046–2059, Dec. 2020.

[25] K. Wang, J. Zhang, G. Bai, R. Ko, and J. S. Dong, "It's not just the site, it's the contents: Intra-domain fingerprinting social media websites through CDN bursts," in *Proc. Web Conf.*, 2021, pp. 2142–2153.

[26] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 1928–1943.

[27] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen, “Automated website fingerprinting through deep learning,” in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018.
[28] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt, “A critical evaluation of website fingerprinting attacks,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 263–274.
[29] P. Dubroy and R. Balakrishnan, “A study of tabbed browsing among mozilla firefox users,” in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 2010, pp. 673–682.
[30] J. Huang and R. W. White, “Parallel browsing behavior on the web,” in *Proc. 21st ACM Conf. Hypertext Hypermedia*, 2010, pp. 13–18.
[31] H. Cheng and R. Avnur, “Traffic analysis of SSL encrypted web browsing,” *Project Paper*, Univ. Berkeley, 1998.
[32] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu, “Statistical identification of encrypted web browsing traffic,” in *Proc. IEEE Symp. Secur. Privacy*, 2002, pp. 19–30.
[33] X. Gong, N. Borisov, N. Kiyavash, and N. Schear, “Website detection using remote traffic analysis,” in *Proc. Int. Symp. Privacy Enhancing Technol. Symp.*, 2012, pp. 58–78.
[34] Z. Zhuo, Y. Zhang, Z.-L. Zhang, X. Zhang, and J. Zhang, “Website fingerprinting attack on anonymity networks based on profile hidden Markov model,” *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 5, pp. 1081–1095, May 2018.
[35] P. Sirinam, N. Mathews, M. S. Rahman, and M. Wright, “Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1131–1148.
[36] M. Liberatore and B. N. Levine, “Inferring the source of encrypted HTTP connections,” in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, 2006, pp. 255–263.
[37] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, “Website fingerprinting in onion routing based anonymization networks,” in *Proc. 10th Annu. ACM Workshop Privacy Electron. Soc.*, 2011, pp. 103–114.
[38] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, “Effective attacks and provable defenses for website fingerprinting,” in *Proc. 23rd USENIX Conf. Secur. Symp.*, 2014, pp. 143–157.
[39] M. S. Rahman, P. Sirinam, N. Mathews, K. G. Gangadhara, and M. Wright, “Tik-tok: The utility of packet timing in website fingerprinting attacks,” *Proc. Privacy Enhancing Technol.*, vol. 3, pp. 5–24, 2020.
[40] Internet censorship and surveillance by country, 2021. Accessed: Mar. 22, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Internet_censorship_and_surveil-lance_by_country,note=
[41] K. Garimella, T. Smith, R. Weiss, and R. West, “Political polarization in online news consumption,” in *Proc. Int. AAAI Conf. Web Soc. Media*, 2021, pp. 152–162.
[42] Hypertext Transfer Protocol Version 2 (HTTP/2), 2015. Accessed: Mar. 22, 2022. [Online]. Available: https://tools.ietf.org/html/rfc7540
[43] Hypertext Transfer Protocol Version 3 (HTTP/3), 2020. Accessed: Mar. 22, 2022. [Online]. Available: https://tools.ietf.org/html/draft-ietf-quic-http-32
[44] G. Mitra, P. K. Vairam, P. Slpsk, N. Chandrachoodan, and V. Kamakoti, “Depending on HTTP/2 for privacy? Good luck!,” in *Proc. IEEE/IFIP 50th Annu. Int. Conf. Dependable Syst. Netw.*, 2020, pp. 278–285.
[45] M. Trevisan, S. Traverso, E. Bassi, and M. Mellia, “4 years of EU cookie law: Results and lessons learned,” in *Proc. Int. Symp. Privacy Enhancing Technol. Symp.*, 2019, pp. 126–145.
[46] M. Husák, M. Čermák, T. Jirsík, and P. Čeleda, “HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting,” *EURASIP J. Inf. Secur.*, vol. 2016, no. 1, pp. 1–14 2016.
[47] M. Laštovička, S. Špaček, P. Velan, and P. Čeleda, “Using TLS fingerprints for OS identification in encrypted traffic,” in *Proc. IEEE/IFIP Netw. Operations Manage. Symp.*, 2020, pp. 1–6.
[48] Implementation of existing webpage fingerprinting techniques, 2019. Accessed: Mar. 22, 2022. [Online]. Available: https://github.com/hfalan/codaspy19
[49] M. Shen, Y. Liu, S. Chen, L. Zhu, and Y. Zhang, “Webpage fingerprinting using only packet length information,” in *Proc. IEEE Int. Conf. Commun.*, 2019, pp. 1–6.
[50] Open this story in a new tab, 2010. Accessed: Mar. 22, 2022. [Online]. Available: https://slate.com/human-interest/2010/12/a-new-data-set-from-firefox-reveals-our-browsing-habits.html
[51] J. Gong and T. Wang, “Zero-delay lightweight defenses against website fingerprinting,” in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 717–734.
[52] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, “Peek-a-Boo, I still see you: Why efficient traffic analysis countermeasures fail,” in *Proc. IEEE Symp. Secur. Privacy*, 2012, pp. 332–346.
[53] Z. Ling, G. Xiao, W. Wu, X. Gu, M. Yang, and X. Fu, “Towards an efficient defense against deep learning based website fingerprinting,” in *Proc. IEEE Conf. Comput. Commun.*, 2022, pp. 310–319.

Gargi Mitra received the BTech degree in computer science & engineering from St. Thomas’ College of Engineering & Technology. She is currently working toward the PhD degree in the Interdisciplinary Research Program (CSE-EE) with IIT Madras. Her areas of research interest include Internet security and privacy.

Prasanna Karthik Vairam received the MTech degree from IIT Bombay, and the PhD degree from IIT Madras. He is a lecturer with the School of Computing, National University of Singapore. His research interests include blockchains, approximate data structures, and systems security.

Sandip Saha received the BTech degree in computer science & engineering from Jalpaiguri Government Engineering College. He is currently working toward the MS degree with IIT Madras. His research interests include internet security.

Nitin Chandrachoodan (Member, IEEE) received the BTech degree in electronics and communication engineering from IIT Madras, in 1996, and the PhD degree from the University of Maryland, College Park, in 2002. He is an associate professor with IIT Madras. His research interests include digital systems design and security. He is also an associate editor of the *Journal of Signal Processing Systems (Springer)*.

V. Kamakoti is a professor and the director of IIT Madras. He specializes in the areas of computer architecture, secure systems engineering, and network security and privacy. He is a coordinator of the Information Security Education and Awareness program of the Department of Information Technology, Government of India and the chairman of the Task Force on Artificial Intelligence for India’s Economic Transformation. He has also won several awards such as the IBM Faculty Award (2016).