Deep Inside Tor: Exploring Website Fingerprinting Attacks on Tor Traffic in Realistic Settings

Conference Paper · November 2022

DOI: 10.1109/ICCKE57176.2022.9960104

Amirhossein Khajehpour
Sharif University of Technology
7 PUBLICATIONS **15** CITATIONS
SEE PROFILE

Navid Malekghaini
University of Waterloo
5 PUBLICATIONS **13** CITATIONS
SEE PROFILE

Naeimeh Omidvar
Institute for Research in Fundamental Sciences (IPM)
21 PUBLICATIONS **108** CITATIONS
SEE PROFILE

Mahdi Jafari Siavoshani
Sharif University of Technology
59 PUBLICATIONS **1,223** CITATIONS
SEE PROFILE

The paper has been accepted for publication in 12th International Conference on Computer and Knowledge Engineering (ICCKE 2022), November 17-18, 2022, Ferdowsi University of Mashhad, Iran. This is an author's copy, the respective copyrights are with IEEE.

Deep Inside Tor: Exploring Website Fingerprinting Attacks on Tor Traffic in Realistic Settings

Amirhossein Khajehpour*†, Farid Zandi*†, Navid Malekghaini†, Mahdi Hemmatyar†, Naeimeh Omidvar‡, Mahdi Jafari Siavoshani†

†*Department of Computer Engineering, Sharif University of Technology, Tehran, Iran*

‡*School of Computer Science, Institute for Research in Fundamental Sciences (IPM)*

{amirhosseinkh, zandish, nmalek, hemmat}@ce.sharif.edu, {omidvar}@ipm.ir, {mjafari}@sharif.edu

***Abstract*—In recent years, with the new advances in the areas of machine learning, Tor's advertised anonymity has been widely threatened. Despite all the protection mechanisms employed by Tor, attackers can now draw inferences on the online activities of a Tor user. Although such an study is critical for users of Tor, most of the existing works in this regard are based on unrealistic assumptions and settings. In this work, we explore the effectiveness of fingerprinting attacks under realistic settings. We focus on identifying the target websites and applications visited or used by a Tor user, through analyzing the heavily encrypted traffic that any local eavesdropper can also see. Unlike existing works, we focus on small groups of consecutive packets, which allows us to study more complex user behavior. By modifying our Tor client to label the Tor cells with the name of the destined application or website name, could label the packets even when multiple websites and applications were simultaneously using the Tor proxy. To label the network packets, the byte sequence of a labeled cell was located inside the packets, which, to the best of our knowledge, is the most accurate way for labeling the packets. In this way, we accomplished to gather an extensive dataset of Tor traffic, corresponding to different types of user's behavior. Finally, we proposed several deep neural network structures to classify the packets of different websites and analyzed the effectiveness of Tor's encryption methods in realistic settings by achieving an accuracy of 3% in classifying 100 different websites. The results show the effectiveness of Tor's multi-layer encryption scheme.**

1. Introduction

Tor [1] is an anonymity network designed to anonymously route the internet traffic of its clients through a worldwide network of relay nodes using multiple layers of encryption. It aims at providing its clients with an opportunity to anonymously use the internet, by hiding their activities from oppressive regimes, malevolent ISPs, etc. Specifically, its low latency makes it suitable for interactive activities such as web browsing. This advantage, however, comes at the cost of making it impossible for Tor to hide all the information about its users' activities. In this regard, some recent works [2–11] show that any local eavesdropper (who can observe the internet traffic between the user and the first node of the Tor network) can indeed recognize the websites that a Tor user is visiting. Such an attack is commonly known as the *website fingerprinting attack*.

To conduct the website fingerprinting attack, the adversary eavesdropper connects to the Internet through the anonymity network (Tor in our case), and visits the set of the websites they wish to monitor. The resulting packet traces are then captured and investigated by the adversary eavesdropper to develop a learning model for classifying the incoming packets of the Tor networks for the websites of interest. This process is done repeatedly until the attacker learns how to accurately classify different packet traces according to their originating websites (the training phase). After the training phase, the attacker sniffs the user's traffic, and then using the trained model, they try to identify which website the user is visiting (the testing phase).

Up until recently, it was widely believed that no information concerning the user's activity would leak from encrypted traffic. Surprisingly, though, some recent works have shown that this might not always be the case. For example, [12] proposed a novel fingerprinting attack and showed that an eavesdropper, with considerable accuracy, can extract undisclosed information from the encrypted payload in TLS protocol.

In this work, we aim at verifying that Tor's encryption system is safe against such attacks in a real-world environment. In order to accomplish this goal, a dataset of encrypted payload transmitted through Tor network would be neces-

*. Indicates equal contribution.

sary. However, previous works that have gathered datasets for similar purposes have not published the encrypted payload of their packets [2–5, 7–10]. Additionally, their works are based on two major assumptions on the victim's online behaviour: 1) only one page would be loaded at a time; and 2) no background task is generating any extraneous traffic other than the main traffic [6]. Such assumptions are generally invalid in many scenarios, making their attacks unrealistic or impractical.

To address the aforementioned issues of the existing datasets and collect a more general and realistic Tor traffic dataset, in this work, we classify the Tor packets individually based on the encrypted traffic, which would allow us to conduct the fingerprinting attack in a general and practical scenario, where multiple browsers or applications could be simultaneously working, and also where background noise may be present. This approach is in contrast to the previous works, which use the entire packet trace of the page loading procedure to classify the traffic. It should be noted that, in the multi-session and noisy setting described above, it is quite challenging to label packets according to their originating websites and applications. This is due to the fact that packet traces of various sources would be intertwined together in this situation, and hence, labeling the training data would be non-trivial. This type of attack has not formerly been studied. The main contributions of this work can be summarized as follows:

- We proposed a novel approach for labeling Tor network packets, by modifying our Tor client [1] and OpenSSL [2] library. This is, to the best of our knowledge, the most accurate approach for labeling the training data.
- We collected an extensive dataset consisting of the Tor-generated network traffic resulted from crawling 100 domains. To gather the corresponding traffic, various scenarios are considered, as follows. We first run our crawlers one at a time, and then run multiple crawlers simultaneously to imitate a more complicated user behaviour. Moreover, we first configured our crawlers to crawl only the homepage of each domain (*Webpage Fingerprinting* attack), and then configured them to include the inner pages as well (*Website Fingerprinting attack*). Table 1 describes each of the collected datasets. P-SinHom, P-MulHom, P-SinAll, and P-MulAll, indicated in Table 1, correspond to the datasets described above. We also collected a dataset of packet traces similar to previous works for comparison and verifications of their results, named as P-SinHom-Bi dataset in Table 1.
- Moreover, we also gathered the traffic generated by the applications whose connections has been tunneled through the Tor network. For this purpose, we automated 10 desktop applications and recorded their labeled traffic (P-MulApp dataset in Table 1).
- Furthermore, we explored the applicability of fingerprinting on Tor users' encrypted data, in case of having adversarial nodes that work either as entry or middle relay nodes inside the victim's circuit. For both scenarios, we gathered cell payload data and tested the vulnerability of Tor's onion skins encryption at each point along the circuit against the fingerprinting attack. We named this new type of attack as *Cell Fingerprinting.*
- Finally, using both convolutional deep neural network, and random forest we conduct fingerprinting attacks on our gathered datasets. As verified by various experimental results, the classification accuracies in the conducted tests hardly exceeds those of the random classifier. This indicates that Tor's encryption is indeed safe againts the state-of-the-art deep learning methods.

The rest of the paper is organized as follows: Section 2 reviews the related works in the literature. Section 3 provides a brief background on Tor. Section 4 presents the details of our fingerprinting attacks and dataset capturing process. The specifications of our collected datasets is explained in section 5. Section 6 discusses the proposed deep learning methods for classification of packets and the experimental results. Finally, Section 7 concludes the paper.

1. https://github.com/INL-Laboratory/Tor-Traffic-Labeling
2. https://github.com/INL-Laboratory/RawTLSRecordsOpenSSL

2. Related Work

The first successful website fingerprinting attack on the Tor network was performed in 2011 by Panchenko et al. [2]. Unlike our work which focuses on classification of individual packets, this work and other previous works use entire packet traces to classify the traffic. Working with traces allowed them to use features such as total traffic size in each direction and the percentage of the incoming bytes. They also placed markers indicating the direction change in packets. Using an SVM classifier for fingerprinting 755 web pages, they could dramatically increase the accuracy to 55%, compared to 3% in earlier works. Moreover, their work was the first to consider an open-world setting (where in addition to the monitored set, there is also a set of unmonitored websites that the attacker does not wish to identify), and could reach a 57% True Positive Rate (TPR) while monitoring five pages. Later in 2012, Cai et al. used features derived from the optimal string alignment distance (OSAD) of the network traces, and could attain a 70% accuracy in fingerprinting 800 pages [3].

In 2013, Wang and Goldberg considered the Tor cells, instead of packets, as the unit of transmission, and based on that, they tried to statistically remove the SENDME cells [13]. This helped them to achieve 91% accuracy for fingerprinting 100 websites. Later on, in 2014, they could further improve their accuracy to 95%, employing a K-Nearest Neighbors (KNN) classifier on roughly 4000 features extracted from the packet traces [4, 5]. Next, in 2016, they tried to verify their findings in a more realistic environment, and examined the effects of introducing background noise and loading multiple pages at the same time [6]. Their attempts at removing the noise showed a highly inferior accuracy. Moreover, in the case of loading multiple pages, they first split the cell trace into segments, wherever there was a gap

#	Dataset	Type	Threads Count	Content	Site/App Count	Size
1	**P-SinHom**	Packets	Single-Thread	Homepage	100 Websites	1.5×10^6 Packet batches
2	**P-MulHom**	Packets	Multi-Thread	Homepage	100 Websites	1.5×10^6 Packet batches
3	**P-SinAll**	Packets	Single-Thread	Homepage+Inner Pages	100 Websites	1.5×10^6 Packet batches
4	**P-MulAll**	Packets	Multi-Thread	Homepage+Inner Pages	100 Websites	1.5×10^6 Packet batches
5	**P-MulApp**	Packets	Multi-Thread	Application	10 Applications	1.5×10^3 Packet batches
6	**C-MulHom**	Tor Cells	Multi-Thread	Homepage	100 Websites	10^8 Tor cells
7	**P-SinHom-Bi**	Packet Trace	Single-Thread	Homepage	100 Websites	1.13×10^8 Packets

TABLE 1: Description of our collected datasets.

between the cells with a duration of more than one second. The resulting segments were further split if they concluded that these segments included packets of two different pages. For this purpose, they need to find the splitting point, i.e. wherein the segments stream, the second page started to be loaded. In the end, the segments were used to predict the destined website. Their results showed that when the page loadings overlapped, their accuracy was sharply reduced.

Later works tried to further increase accuracy and reduce learning time and processing power [7, 8]. In 2016, Panchenko et al. presented CUMUL, a fingerprinting approach focusing on the cumulative size of transmitted data through the time when the page is being loaded [7]. Their intuitive approach achieved 92% accuracy for classification of 100 pages. This was the first work to use both inner pages of websites and their homepages for fingerprinting, in contrast to the previous works that only considered the latter. Moreover, unlike most of the earlier works, which used the Alexa top 100 pages, they chose pages from Google trends or some URLs found on Twitter, which they argue that could serve as better representatives for the entire internet traffic. Another work in the same year by Hayes et al. used random forests to extract *fingerprints* for the traces of each website [8]. This approach resulted in a significant reduction in the False Positive Rate (FPR) in an open-world setting. In [9] the authors utilized deep learning methods to automatically extract useful features from the traffic traces. They examined various deep learning models including feedforward, convolutional, and recurrent deep neural networks, and could reach 96% and 94% accuracy for 100 and 900 websites, respectively. Furthermore, in 2018, Xu et. al studied fingerprinting when multiple pages were loaded at the same time [11]. They introduced the *multi-tab threat model*, where two pages started to be loaded within a time gap of 2 to 6 seconds, and proposed an algorithm to accurately find the moment in which the loading of the second page begins. Using this moment, they discarded all the packets that have been recorded since both pages were being loaded, and conducted fingerprinting attack on the remaining packets, with a random forest using the Gini index as their tree index method. Such a fingerprinting attack achieved 64.9% TPR with a dynamic splitting method on their Tor dataset.

It should be noted that some of the previous works have also presented defensive methods against fingerprinting attacks on Tor network, including BuFLO [14], Tamaraw [15], WTF-PAD [16], and Walkie-Talkie [13]. Most of these methods are based on synthesizing traffic, padding it with dummy cells, or extending the packet delays. Some further attacks have also been proposed to break the aformentioned defenses. For example in 2018, Sirinam et al. [10] proposed Deep Fingerprinting attack powered by CNNs which could reach 90% accuracy even when WTF-PAD defense was enabled, and 49.7% accuracy against Walkie-Talkie. Moreover, in the absence of defensive methods they could attain 98.3% accuracy, outperforming the previous works.

Finally in 2022, Jafari et al. proposed a novel fingerprinting attack on SSL-encrypted traffic, which could classify the packets traces for 100 websites with 15.57% accuracy, solely based on the encrypted payload of the network packets [12]. Their findings contradict the common belief that encrypted traffic is completely secure against information leakage. It should be noted that no existing work in the literature has yet utilized the encrypted payload for classification of Tor encrypted traffic.

3. Background on Tor Anonymity Network

To provide secure communication for possibly restricted or sensitive destinations, a Tor client provides SOCKS proxies for user applications. To significantly reduce the traceability of the users' activities, the data is heavily encrypted and sent through paths called *circuits*, which are hard to completely discover for parties other than the Tor client. The traffic flows through the Tor network in units called *cells*. In this paper, we focus on cells of the RELAY type, which contain the applications' transmitted data, and can be used for the fingerprinting attack. Other types of the cells, such as CREATE and EXTEND, are mainly used to configure the circuits in the Tor network.

To anonymously exchange traffic in the Internet, a Tor client requires to create a circuit first, which is a bidirectional path in the Tor network between the client and a Tor relay node called the *exit* node. This path also includes two other nodes called the *entry* and *middle* nodes in between, as depicted in Fig. 1. To build a circuit, the client first selects three nodes as entry, middle and exit nodes. Then, it sends a CREATE cell to the entry node and establishes an SSL-encrypted connection with it. Then, it sends an EXTEND cell to the entry node, informing it of the selected middle node. The entry node will in return, send a CREATE cell to the middle node and connects to it, thus extending the circuit one more step. Finally, the client sends an EXTEND cell to the middle node, informing it of the selected exit node, and the middle node will send a CREATE cell to the exit node, connecting and extending the circuit to it. Note

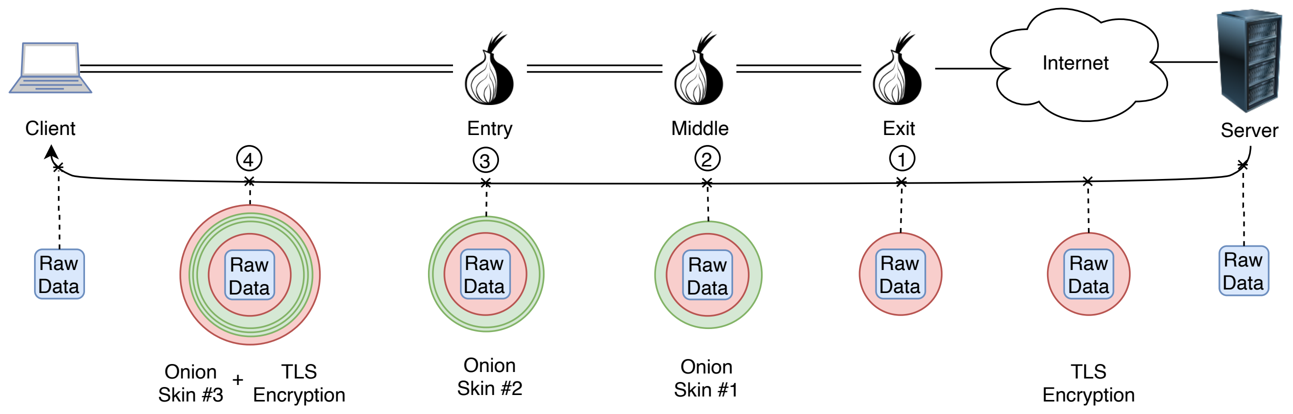

Figure 1: Overview of the relays in a Tor circuit. This circuit constitutes three layers of onion skin, and two TLS encryptions as illustrated by the green circles.

that selection of the nodes in the above protocol is based on various important parameters including their physical location, available bandwidth, and exit policy [1]. Moreover, in each step of the above protocol, a pair of symmetric keys (namely *backward* and *forward* keys) is shared between the Tor client and the selected node, which independently encrypts the corresponding traffic. This constitutes three layers of encryption as illustrated in Fig. 1 by the green circles.

Whenever an application attampts to make a TCP connection to some destination through the Tor network, an appropriate circuit will be selected or a proper one will be created using the aforementioned protocol. Then, instead of the user application, the exit node of the selected circuit will establish the TCP connection with the desired destination. The data associated with the TCP byte stream is sent and received in the form of RELAY cells through the circuit. In the forward direction (i.e., from the client to the exit node), each RELAY cell is first encrypted three times with forward keys corresponding to each node, before being sent by the TOR client. Then, each node decrypts the cell with its own forward key and sends it again toward the exit node. By the time the cell is received and decrypted by the exit node, it is fully decrypted and ready to be sent to the original destination through the previously established TCP connection. For the data sent in the backward direction, which is received by the exit node and sent towards the client, each node encrypts the cell with its backward key before sending it through the circuit. After receiving each cell, the client needs to decrypt it three times to find the original message. Such a multiple-layered encryption structure is widely referred to as the *onion skins* [1].

Under the elaborated protocol and as can be also verified from Fig. 1, a local eavesdropper who is trying to fingerprint the traffic sent from the entry relay node to the client, can only observe a stream of data that has been encrypted multiple times, as follows:

- The original connection between the user application and its destination may be encrypted (e.g SSH connection): The first red circle in Fig 1.
- Corresponding to each relay node of the circuit, there is another level of encryption with AES-CTR-128 cipher: Green circles in Fig 1.
- Connection between each two relay nodes is also SSL-encrypted: The last red circle in Fig 1.

Such a multi-level and hierarchical type of encryption greatly prevents the eavesdropper from inferring any information about the victim's online activity, thus protecting the user from many types of fingerprinting attack.

3.1. Implementation Details of the Tor Source Code

For its TLS connections, Tor uses the OpenSSL library. Each TLS record consists of a 5-byte header, an 8-byte initialization vector (IV) for AES decryption, and a data payload of variable size, as shown in Fig. 2. The data payload is the result of the encryption of the original data, augmented with a 16-byte MAC (therefore, for each received record, OpenSSL would decrypt the data, verify the MAC, and return the decrypted data to Tor). Accordingly, each record is 29 bytes longer than the original data.

In the Tor client source code, the incoming TLS-decrypted data for each connection is stored in an instance of a buffer structure named buf_t. Each buffer is a queue of chunk_t instances that are filled with data and added to tail of the queue, sequentially. When sufficient data has been accumulated in the buffer, an instance of cell_t structure will be created from the head of the queue and its onion skins will be decrypted. The Tor client will finally process the decrypted cells, by using the configuration cells to manage the circuits, and handing the content of the RELAY cells to the user applications. Figure 3 visualizes the data structures inside a buffer.

4. Methodology

In order to train our packet classifier, we first collect the required datasets described in Table 1. We consider two different types of adversaries to conduct the fingerprinting attacks: 1) a local eavesdropper who can observe the network packets between the user and the first relay node of the circuit, and 2) a relay node that tries to fingerprint the traffic of its associated circuit, based on the cells it receives. In the rest of this section, we will illustrate how we gathered a training set for each of these cases.

4.1. Local Eavesdropper

Each packet needs to be labeled with its associated website or application name. Especially, in case of multiple traffic generators working at the same time, it is challenging to accurately determine which generator each packet belongs to, since packets of different traffic generators are mixed together and cannot be split without further information. To overcome this challenge, we decided to label the encrypted Tor cells rather than the packets, and later label the packets using the cells. Accordingly, we modified our Tor client and the OpenSSL library to record two pieces of information for each RELAY cell that is received, as follows:

1) Website or application name: As illustrated before, every RELAY cell received by the Tor client in backward direction is destined to some SOCKS connection. Therefore, to find the destined website or application that each cell belongs to, we find the process that has created such a connection to the SOCKS proxy.[3]
2) The encrypted network traffic corresponding to each Tor cell: We modify the OpenSSL library so as to return the raw TLS records that it reads, in addition to the decrypted data. Note that such records are the traffic that a local eavesdropper might see when monitoring the user's internet connection. We then store these encrypted records alongside the decrypted data in the buffers. Finally, whenever Tor reads some data from the buffer and write it into a cell, we store the corresponding TLS-encrypted data as well. Note that we attribute the header of each TLS record to the nearest possible cell.

Afterwards, we search through the packets for the encrypted payload of the cells, and upon finding the exact byte sequence of a cell's encrypted data in a network packet, we

3. Note that we launched an automated browser for each domain with a unique process name, so that we could separate cells coming from different domains based on their process names.

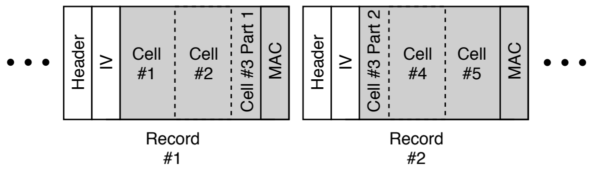

Figure 2: Structure of the TLS records.

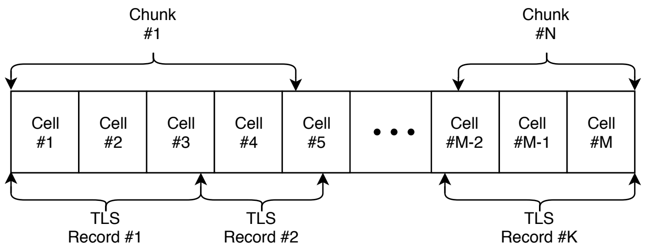

Figure 3: Visualization for data structures inside a buffer. Note that TLS records may include multiple cells and cells might be included in multiple records. Chunks may similarly contain multiple cells waiting to be processed and consequently removed from the chuck.

label the packet with the same label as its corresponding cell. Note that since each packet may contain multiple cells, it might receive multiple labels. Furthermore, a cell could be broken into two packets, as it might start in one packet and end in another one. In those situations, both of the packets that contain the cell will be labeled (see Fig. 3). It's worth mentioning that since the ordering of the cells and their corresponding packets are almost the same, the searching process will not be computationally heavy.

4.2. Relay Nodes

If the adversary is one of the circuit's relay nodes, it will encounter fewer levels of encryption. In particular, as can be seen in Fig. 1, the entry, middle and exit nodes face two, one, and zero levels of encryption over the original application data, respectively. Note that clearly, since the exit node already knows where each cell has come from, there is no point for it in fingerprinting the traffic. The other two, however, may have incentives to find the origin of the packets. Fortunately, to train our classifier, there is no need to deploy any relay nodes to collect and record the cells observed by the entry and middle node. This is because, as described in Section 3, the client decrypts the onion skins one by one, and as a result, it can observe what each relay node of the circuit would have seen. It should be noted that unlike the previous case (i.e., the case of a local eavesdropper), recording the payload of the cells does not involve modifying the OpenSSL library, and can easily be performed by recording the payloads after each level of decryption. Finally, the process of finding the application or website name is exactly the same as the previous part.

5. Dataset

We modified Tor version 0.3.5.0-alpha-dev and OpenSSL version 1.1.1c , according to the modifications illustrated in the previous section and without any interference with the main functionalities of the application. We collected our datasets using two Ubuntu 16.04-powered devices. We capture the packets received on our network interface, using the "tshark" tool. Moreover, we restart our Tor client in certain time intervals to force it to create

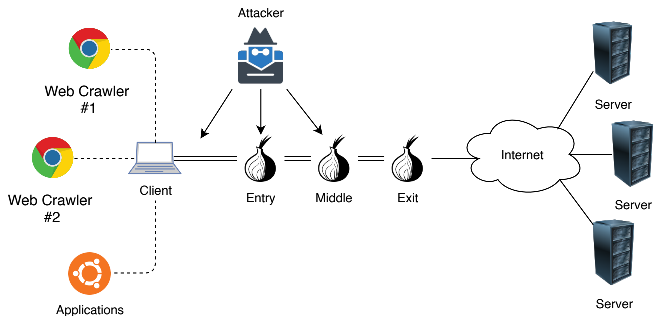

Figure 4: Our setting for collecting the traffic of the web crawlers and the automated applications.

new encryption keys as well as new circuits with different nodes. Furthermore, we turn off the UseEntryGuards option in our Tor client so as to prevent it from choosing the same limited number of entry nodes for circuit creation.

We observe that with high probability, consecutive packets belong to the same source, even in the cases where multiple traffic generators were working at the same time. Therefore, instead of each individual packet, we employ a small number of consecutive network packets as our classification unit, during both training and testing phases. This provides us with much more information to base our decisions upon in comparison with the case when utilizing individual packets. We call such a group of packets a *packet batch*. The size of a packet batch should be large enough to encompass as many packets as possible, and at the same time, small enough not to contain too many packets from different sources. Moreover, we would label a packet batch only if there is at least one source whose proportion of the packets included in that packet batch is higher than a certain threshold. If no such source is found (which might happen as a result of the background noise), we would label the packet batch as *unknown*.

To determine the appropriate values for the packet batch size and the labeling threshold, we first run four crawlers for 20 hours of simultaneous website crawling, and then, calculated the ratio of the *known* batches to all batches for different pairs of the the packet batch size and the labeling threshold. The derived ratios are presented in Table 2. According to this table and in order to guarantee at least 80% labeled packets, we set the packet batch size and the labeling threshold parameters to 23 and 0.5, respectively. Note that a labeling threshold of 0.5 means that labeling a packet batch as known would require that at least 50% of its packets to have come from a certain source. Finally, we gathered the traffic from website crawlers and applications, as will be explained in the rest of the section. Fig. 4 shows our setting for this regard.

5.1. Websites Traffic Datasets

To generate website traffic, a selenium-based crawler was developed that would surf through the pages of a domain, while being connected to the SOCKS proxy of our Tor client. By changing the following two important factors,

	0.2	0.3	0.4	0.5	0.6	0.7	0.8
16	0.920	0.911	0.874	0.803	0.742	0.620	0.556
17	0.922	0.901	0.883	0.823	0.714	0.653	0.533
18	0.924	0.906	0.868	0.799	0.744	0.621	0.516
19	0.925	0.910	0.877	0.818	0.713	0.592	0.500
20	0.922	0.901	0.861	0.797	0.678	0.574	0.485
21	0.923	0.904	0.871	0.816	0.705	0.600	0.513
22	0.924	0.910	0.880	0.793	0.670	0.581	0.500
23	0.926	0.912	0.865	0.809	0.695	0.565	0.485
24	0.927	0.905	0.873	0.784	0.671	0.588	0.472
25	0.923	0.908	0.860	0.791	0.653	0.572	0.460
26	0.924	0.910	0.868	0.777	0.681	0.556	0.485
27	0.926	0.904	0.875	0.793	0.662	0.578	0.471
28	0.927	0.907	0.862	0.775	0.693	0.565	0.461
29	0.928	0.911	0.870	0.789	0.674	0.549	0.449
30	0.925	0.903	0.855	0.772	0.656	0.535	0.436

TABLE 2: Ratio of the *known* batches for each pair of packet batch size and threshold. Vertical axis denotes the packet batch size, and the horizontal axis denotes the labeling threshold level. The ratio values have been calculated based on 20 hours of website crawling by four crawlers working at the same time.

we gathered four datasets representing different types of a Tor user's behaviour.

1) Visit one website at a time or multiple websites, simultaneously. For the first case, we run one crawler for 30 seconds. For the second case, we run four crawlers simultaneously, each assigned with one random website. In that case, different combinations of the websites were considered, and the crawlers were given 120 seconds to visit the pages.
2) Visit just the homepage of each website or the inner pages as well. In the former case, we made our crawler visit the homepage, close the browser, reopen a new instance, and visit the homepage again. This process continue until the considered time period finishes (30 second for one crawler and 120 seconds for 4 crawlers). In the latter case, the links in the visited pages were added to a database (DB), and then is randomly selected by the crawler to visit new pages.

By changing the illustrated factors, we developed four different scenarios. In each of these scenarios, we collected roughly 15000 packet batches for each website. Furthermore, for the random selection of the websites, we used the same 100 highly visited websites in [9], except those few of them that blocked Tor or showed CAPTHAs were manually replaced with some other highly visited websites. We named these four datasets as P-SinHom, P-MulHom, P-SinAll, and P-MulAll and their features are summarized in Table 1.

5.2. Applications Traffic Dataset

To capture the dataset corresponding to the applications traffic, we automated 10 applications using "xdotool" program to generate network traffic. The applications are connected to Tor's SOCKS proxy via the "proxychains" tool, to route both DNS and TCP packets. Moreover, to simulate a real user behavior, at each iteration of six minutes duration, we randomly chose three applications and interactively work with one of them while other two were active in the background. The random applications of each iteration were chosen from various types, including multimedia streaming apps, feed readers, cloud storage managers, and social networking clients.

5.3. Tor Cells Dataset

For our proposed *cell fingerprinting* attack, we modified our Tor client to save the encrypted payload of each cell before removing its onion skins during the decryption process (i.e., to capture the C-MulHome dataset). In the course of capturing P-MulHom dataset, we configure our client to also store cells in order to collect our C-MulHom dataset. Each entry of this dataset contains a cell payload and a label. This dataset includes two types of cells; one layer encrypted, and two layers encrypted. This dataset contains nearly 100 million cells for 100 websites.

5.4. Bidirectional Traffic Dataset

Finally, to compare previous works with ours, we gathered another dataset called P-SinHom-Bi that records packet traces with packets in both directions for 100 pages while being connected to Tor network, ignoring any background traffic during the process. It must be mentioned that this dataset is not labeled with our novel labeling approach.

6. Experimental Results

Using our datasets, in this section, we present the experimental results of various fingerprinting attacks to Tor network traffic in order to explore Tor's safety, under realistic settings. First, note that due to the different number of samples collected for each of the websites, a pre-processing phase was performed on all the datasets to equalize the number of samples of each website class. For this purpose, we *down-sampled* the majority classes, by removing enough number of samples to balance the dataset to prevent domination of the majority classes in the learning process [17]. Moreover, we discarded the packets that had no labels.

6.1. Tor Cells Classification

To perform cell fingerprinting attack, we started by analyzing one-layer encrypted cells (as more layers of encryption makes it harder to extract information form the cells), and conducted our experiments on the C-MulHom. We designed and employed the following two ML models for performing website fingerprinting attacks:

- **Multilayer Perceptron (MLP) Model:** First, we employed an MLP model with five layers, including one input layer, three hidden fully connected layers, and one softmax output layer. After each hidden layer, dropout layer with a 0.2% probability was used. Moreover, we used categorical cross-entropy as the loss function, and Adam algorithm [18] as the optimizer. It was observed that after 1000 epochs, the classification accuracy on the training set was at 1.06%. Next, we started to complicate the model by adding upto five hidden layers with a large number of nodes. However, it did not affect the result. This indicates that the fully connected network cannot learn anything from C-MulHom.
- **Convolutional Neural Network (CNN) Model:** Next, we employed a one-dimensional CNN (1D-CNN). Note that a CNN can capture spatial dependencies between adjacent bytes, which is helpful in finding discriminative patterns of different classes, and hence, achieving a decent classification accuracy [19]. Our architecture included input layer, three convolutional layers, followed by one hidden fully-connected layer and one softmax output layer. The convolutional layers used ReLU as the activation function, and consisted of 256, 128, and 64 filters, respectively. Moreover, each of these layers was followed by a max-pooling layer and a dropout layer with 0.2% probability. After 1000 epochs of training the model, the accuracy on the training data and the test data was 1.68% and about 1%, respectively. Therefore, CNN did not fare much better than the MLP.

The above experiments show that both the utilized learning models cannot learn any meaningful pattern from the Tor cells. Accordingly, in contrast to the findings of [12] about TLS protocol, we can conclude that the Tor encryption algorithm is likely to be so powerful that there is no information leakage to obtain a pattern.

6.2. Tor Packets Classification

To test our four datasets against the fingerprinting attack, we start by performing experiments on the less-complicated one, (i.e, P-SinHom). As described in the Section 4.1, each entry in this dataset contains a packet batch (made of 23 packets, their raw payload, their packet lengths and relative time), with a label. We use the raw payload to classify the packets and ignored any metadata. To obtain the raw payload we needed to do a preprocessing on the packets. We discarded the TLS header and first part of TLS body called IV from it because IV is a nonencrypted incremental number and can identify the sequence of several packets. Then, the following ML models were used for classification:

- **MLP & 1D-CNN Model:** The architectures are the same as above, except only for the input layer size. The input is the concatenation of the initial 200-bytes of all the packets in the packet batch, resulting in a 4600-byte string. Additionally, the packets shorter than

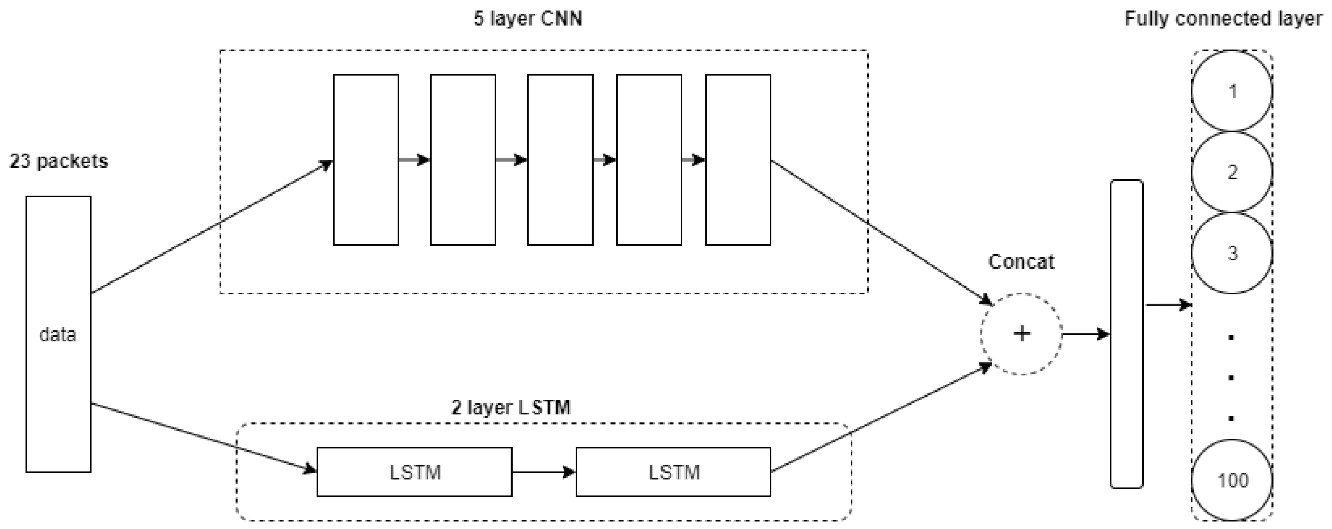

Figure 5: CNN with LSTM used for packets classification.

200 bytes were zero-padded. After 1000 epochs, the testing accuracies are 1.4% and 2.38%, for the MLP and the 1D-CNN models, respectively.

- **2D-CNN Model:** We used a two-dimensional CNN consisting of an input, three 256-filter convolutional, one hidden fully connected, and a softmax output layers. Since we need a two-dimensional matrices for the input of concatenation of packet's initial 200-bytes, we placed the payloads in rows of a 23×200 matrix. The accuracy obtained on the testing data was 1.8%, and that is showing two-dimensional data does not help the model to learn better and only misleads it and reduces its accuracy due to the 1D-CNN.
- **LSTM and CNN Model:** The main idea of this model is taken from the article [20]. This model uses an LSTM and a CNN network, as shown in Fig. 5. We used a 2-layer LSTM to extract time features of each packet's batch, and designed a 5-layer CNN to learn the spatial features of packets. Moreover, we flattened the output of the CNN network and merged it with the output of LSTM network to make a feature vector. Finally, we fed the feature vector to a softmax layer and achieved classification accuracy of 3.23%.
- **Random Forest Model:** We implemented a random forest with 50 trees with depth of the trees increasing from 10 to 90 in steps of 5. It was observed that after the depth of 50, the network was overfitted, as the accuracy on the training set reached 90%while, the accuracy for test set was only at 2%.

6.3. Classification with Payload and Header

As illustrated above, the Tor packet batches could not be successfully classified based on their encrypted payloads solely. Therefore, we decided to also incorporate the TCP *flags*. Note that the flag fields mainly have only *ack* and *push* flags set, and other TCP flags like *syn, fin, urgent*, etc are unset. This experiment yields an accuracy of 10.8% on the test set.

7. Conclusion

In this work, we have explored the idea of fingerprinting Tor traffic using the encrypted payload of the network packets. To this end, we captured extensive datasets and used various methods to conduct the attack. Our results show that Tor's encryption system is immune to such attacks, since we could not attain any acceptable classification accuracy on neither any datasets. Furthermore, the datasets that we gathered for our work is publicly accessible and can be used for further investigations on the Tor network. Specifically, compared to the other available datasets, it benefits from two main advantages, as follows: First, the technique we used for labeling the packets is much more accurate, because of the novel approach that we presented for this purpose, and second, our datasets contain the actual network packets and their encrypted payloads, which is usually summarized or removed in the other works.

References

[1] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," tech. rep., Naval Research Lab Washington DC, 2004.

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

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

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

[5] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for website fingerprinting," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pp. 143–157, 2014.

[6] T. Wang and I. Goldberg, "On realistically attacking tor with website fingerprinting," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 4, pp. 21–36, 2016.

[7] A. Panchenko and F. Lanze, "Website fingerprinting at internet scale.," 2016.

[8] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pp. 1187–1203, 2016.

[9] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen, "Automated website fingerprinting through deep learning," *arXiv preprint arXiv:1708.06376*, 2017.

[10] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1928–1943, ACM, 2018.

[11] Y. Xu, T. Wang, Q. Li, Q. Gong, Y. Chen, and Y. Jiang, "A multi-tab website fingerprinting attack," in *Proceedings of the 34th Annual Computer Security Applications Conference*, pp. 327–341, ACM, 2018.

[12] M. Jafari Siavoshani, A. H. Khajepour, A. Ziaei, A. A. Gatmiri, and A. Taheri, "Machine learning interpretability meets tls fingerprinting," *arXiv preprint arXiv:2011.06304*, 2020.

[13] T. Wang and I. Goldberg, "Walkie-talkie: An efficient defense against passive website fingerprinting attacks," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pp. 1375–1390, 2017.

[14] 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 *2012 IEEE symposium on security and privacy*, pp. 332–346, IEEE, 2012.

[15] X. Cai, R. Nithyanand, and R. Johnson, "Cs-buflo: A congestion sensitive website fingerprinting defense," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pp. 121–130, ACM, 2014.

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

[17] F. Provost, "Machine learning from imbalanced data sets 101," in *Proceedings of the AAAI'2000 workshop on imbalanced data sets*, vol. 68, pp. 1–3, AAAI Press, 2000.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[19] N. Malekghaini, E. Akbari, M. A. Salahuddin, N. Limam, R. Boutaba, B. Mathieu, S. Moteau, and S. Tuffin, "Data drift in dl: Lessons learned from encrypted traffic classification," in *2022 IFIP Networking Conference (IFIP Networking)*, pp. 1–9, 2022.

[20] X. He, J. Wang, Y. He, and Y. Shi, "A deep learning approach for website fingerprinting attack," in *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, pp. 1419–1423, IEEE, 2018.