University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Cybersecurity

Divergent: nothing as it seems. A novel defence against Website Fingerprinting attacks

Supervisor
Prof. Mauro Conti
University of Padova

Co-supervisor
Dr. Chhagan Lal
TU Delft

Master Candidate
Francesco Varotto

Student ID
2022814

Academic Year
2021-2022

"Stand on the shoulders of giants"
— Isaac Newton

Abstract

Website Fingerprinting (WF) attacks exploit Network Side Channels (NSCs) in order to obtain user's web activity, leak secrets or identify the requested web page. The means through which a malicious user could exploit are, among all, packet timing, packet sizes and traffic shape, even when the communication channel is encrypted or anonymized. Specifically, he gathers network traffic generated while a user accesses a website, and then exerts a series of techniques to discover patterns of the network flow to infer the type of website the victim inquires. State-of-the-art WF attacks have been shown to be effective even against privacy technologies that have as main goal to protect and hide the identity of the users during their network activities (such as Tor, VPNs). These threats are of particular concern since break the privacy, and the anonymity, expected by users who employ such frameworks. With our proposal, we explore a new method for improving the flaws that distinguish the principal defences that have been presented, while still maintaining good performances against WF attacks. The other defences suppose to modify the network protocol, pair each web page to a decoy one or demand too much bandwidth.

In this thesis we present a traffic analysis attack for WF, that leverages a deep learning's model called Convolutional Neural Networks (CNN), in both traditional and anonymity networks (especially against Tor). With CNN, the attacker is able to identify individual pages in the same website with more than 92% and 95% of accuracy, respectively for traditional and Tor networks. Then, we propose a novel defence, named Divergent, which is capable of reducing the impact of the attack. Our countermeasure lowers the confidence of the output of the adversarial model, introducing only 23% of bandwidth overhead and almost 0% of time overhead on average. Divergent is based on the idea of changing the traffic fingerprint strictly tight to a resource upon each client's request, leveraging randomness and dummy packets.

Contents

Abstract v

List of figures ix

List of tables xi

Listing of acronyms xiii

1 Introduction 1

2 Background 5
2.1 Traffic analysis attacks 5
2.2 Tor network 6
2.3 Deep Learning and CNNs 8

3 Related work 11

4 Identifying resources using Neural Networks 15
4.1 Adversarial CNN-model 15
4.2 Data collection & features exploited 17

5 Our proposal - Divergent 19
5.1 Threat model 19
5.2 Our countermeasure: Divergent 21
5.3 Bandwidth Overhead and Time Overhead 27
5.4 Importance of randomness 30

6 Performance evaluation 33
6.1 Evaluation setup 33
6.2 Evaluation metrics 34
6.3 Results 34

7 Discussion & future work 37

8 Conclusion 41

References 43

Acknowledgments 47

Listing of figures

2.1 Tor network with local eavesdropper 7
2.2 Layered Tor cell 8
2.3 Example of fully-connected Neural Network 9

3.1 Idea Walkie-Talkie 12

4.1 Adversarial Convolutional Neural Network architecture 16

5.1 System models in the two different scenarios 21
5.2 Plot of two binomial distributions 24
5.3 Example scenarios without (left) and with (right) Divergent 25
5.4 Construction of dummy packet with random (top) and xored (bottom) payload 26
5.5 Bernoulli distribution implemented in our PC 32

6.1 Comparison confusion matrices without Divergent (left) and with defence (right) 36

Listing of tables

3.1 WF defences. 14

4.1 Exploited features in the traditional network setting and relative description. . 18
4.2 Exploited features in the Tor network setting and relative description. 18

5.1 Different scenarios for time overhead 29
5.2 Bandwidth (BWOH) and Time (TOH) Overhead of some of the WF defences. 30

6.1 Output probabilities of the adversarial CNN model against five different videos with Divergent 36

Listing of acronyms

CNN . . . . . Convolutional Neural Network

DL . . . . . Deep Learning

GPU . . . . . Graphics Processing Unit

HTTP . . . . . Hypertext Transfer Protocol

LAN . . . . . Local Area Network

LoL . . . . . League of Legends

MiTM . . . . . Man in The Middle

ML . . . . . Machine Learning

MTU . . . . . Maximum Transfer Unit

NSC . . . . . Network Side-Channel

OR . . . . . Onion Router

PMF . . . . . Probability Mass Function

QoE . . . . . Quality of Experience

ReLU . . . . . Rectified Linear Unit

SVM . . . . . Support Vector Machine

tanh . . . . . Hyperbolic Tangent

TLS . . . . . Transport Layer Security

VPN . . . . . Virtual Private Network

WF . . . . . Website Fingerprinting

1 Introduction

Privacy and confidentiality are two cutting-edge topics that depict our contemporary digital age and for this reason researchers develop and improve cryptographic protocols day-by-day [1, 2, 3, 4]. Unfortunately, the gap between the practical implementations of such protocols, such as TLS, which is the de-facto standard for HTTPS, and the theoretical context in which they are actually considered, opens the potential for side channel attacks. A side-channel attack is a security exploit, which attempts to extract secrets, which are leaked by these particular *side channels*, that are intrinsic in a system, or in a protocol, due to its implementation. For example, Compagno A. et al. [5] exploit a keyboard acoustic eavesdropping attack, in order to extract information coming from VoIP calls (especially, Skype) or Schuster R. et al. [6] managed to recognize the video resources requested by a client, analyzing TCP packets over an encrypted channel.

Traffic analysis attacks are a subset of NSC attacks and their effectiveness deeply depends on both the classifier algorithm and the set of features employed. Previous offensives leverage, mainly, hand-crafted features in order to represent the network traffic, and exploit classifiers such as Support Vector Machine (SVM) [7], k-Nearest Neighbours (k-NN) [8] or random forests [9]. With these classifiers the attacker is able to achieve 90% of accuracy. Later, in 2018, Sirinam P. et al. [10] propose a new classifier, which is based on the Deep Learning (DL) model named Convolutional Neural Network (CNN). Thanks to this innovative approach, DL outperforms traditional machine learning techniques, reaching more than 98% of accu-

racy against Tor traffic without defence and 49.7% against one of the best proposed defence, called Walkie-Talkie [11].

Prior works have shown that there exist several efforts, which try to bypass such malicious attempts [12, 13, 14, 15, 11]. WF defences try to harden and hinder the main features exploited by the attackers, including packet length and arrival time, using different techniques, such as fake packets and delay. Despite the progress in lowering the performances of the adversarial, these proposals introduce a lot of overhead, make strict assumptions on the client (such as she must know in advance the characteristics of the resource she is asking for), or are not efficient (e.g., besides the overhead, propose to pair each web-page with a fake one).

The main contribution of this thesis is to address such disadvantages. In order to do so, we propose a novel bypass against a specific type of traffic analysis attack, called website fingerprinting. WF attacks allow a curious attacker, who controls a device in the same network of the victim, to gain information about the encrypted resources transmitted by the server. Our idea rests on continuously altering the traffic flow of the same resource requested by a client, leveraging randomness and dummy packets.
The principal properties of our proposal are the following ones:

- *Robust*, since it is capable of bypassing website fingerprinting attacks, which leverage timing and packet ordering, besides packet length;
- *Efficient*, as it is based on the idea of modifying the traffic flow of the same data upon different client's requests, and on exploiting, eventually, the payload of dummy packets, it does reduce the overhead introduced;
- *Ease of deployment*, seeing as how our protocol does not impact web servers, the only change is applied on the application layer.

The thesis is organized as following: in chapter 1 we give an introduction to the main subjects of this work, including some references to side-channel attacks and some WF attacks and defences. In the following chapter we analyze in depth WF attacks, how the Tor network is structured and works and finally we briefly examine the general world of Deep Learning (DL) and one of its models, named Convolutional Neural Network (CNN). In chapter 3 we discuss some of the WF defences that have been proposed, their main ideas and limitations. Chapter 4 deals with our implementation of the adversarial attacker, namely the offensive CNN model and the exploited features. Following, our defence is presented and deeply described,

considering the overhead introduced, too. In chapter 6 we evaluate Divergent against our implementation of the attacker. In chapter 7 we discuss some limitations and improvements that can be taken into account in order to improve our proposal. Finally, in chapter 8 we conclude the thesis.

2 Background

In this chapter we provide the fundamental notions of traffic analysis attacks, with a focus on the website fingerprinting ones, followed by what the Tor network is and how it works. Furthermore, we are going to discuss about the emerging and important field of Deep Learning and one of its development, resulting in CNNs.

2.1 Traffic analysis attacks

Similar to eavesdropping, traffic analysis attacks are passive and based on what the adversarial is able to grasp from the network. However, differently from an active eavesdropping attack (e.g., MiTM), in this kind of offensive the attacker doesn't compromise data.
The website fingerprinting (WF) attack is a peculiar example of traffic analysis. Performed by a local, passive eavesdropper, it aims to infer information about the content (i.e., the website visited or the requested resource) of encrypted and/or anonymized connections, by observing patterns of traffic data. Here, the attacker merely utilizes meta information, such as packet size and direction or shape of traffic, without breaking the encryption mechanisms of the channel. Typically, the WF problem is treated as a supervised classification problem, where the possible classification categories are URLs, and network traffic traces, namely sequences of packet inter-arrival times and packet lengths, are the observations. To deploy the attack, the adversary first trains a classifier with network traffic traces collected from his own visits to a set of target web pages he later wants to identify and link to users. Next, the attacker records and classifies user

traffic and uses it to make a guess of the page, or data, that could have been accessed.
Before Panchenko et al. published a research [13] at the end of 2011, in which they were able to achieve a good accuracy level with a WF attack against Tor, Tor was considered to be secure against this threat. Since then, it has become an active field of research and several related works showed the feasibility of the WF attack in Tor and against other mechanisms and infrastructures. For example, in 2017 Schuster R. et al. [6] published a paper in which they explored an attack which was able to identify, from an encrypted flow of data, which streaming video a targeted user was watching from specific services, due to the leakage of information caused by the MPEG-DASH video standard, which has become the de-facto standard of video streaming adopted by the major companies, such as YouTube or Netflix. One year later, Jiaxi G. et al. [16] proposed a similar offensive.
Concerning in terms of WF attacks against Tor, on 2014 Wang et al. [8] proposed a new approach in order to notably improve the attacker's performances, specifically, employing a k-Nearest Neighbour classifier, which is also a supervised machine learning algorithm. This ML algorithm takes as input k (number of neighbours) and the output-label is assigned on the basis of a majority vote, i.e. the label that is most frequently represented around a given data point. Note that the value k defines how many neighbors will be checked to determine the classification of a specific query point. This has been one of the classifier with the highest accuracy known [7], until 2018, when Sirinam P. et al. [10] presented a new attack based on Deep Learning models (CNN architecture), which was able to attain over 98% of accuracy on Tor traffic without defenses. Indeed, the implementation of our attack is based on a CNN model.

2.2 Tor network

Tor (The Onion Router) is a free, open-source software launched on 2002 and it is considered the most popular anonymization network, which has more than two million daily users [17]. One of Tor's main goals is low-latency in different applications, such as web browsing.
In its default and standard version, Tor routes connections through virtual tunnels, called circuits, which typically consist of three onion routers (OR), also called nodes. In figure 2.1 we can observe the *Guard*, the *Middle* and the *Exit* nodes.
The traffic is encrypted in layers, namely like an onion, that is the client shares different symmetric encryption keys with each OR on the circuit, encrypts the data with all keys consecutively, and each OR decrypts its layer on the path.
In figure 2.2, the final cell is the result of the encryption of three different symmetric keys, and

the final payload (the *Message* in the figure) is in plaintext only once all the three decryption mechanisms have been taken place, with their respective keys. Observe that the exit node peels off the final layer of Tor's encryption, and sends the fully decrypted payload to the remote server on the other end of the Tor circuit. Hopefully, the client and the server are using a further layer of encryption so that not even the exit node can read their actual plaintext messages. But this is Tor's out of scope. This technique ensures that no relay on a path can know both the origin and destination of a transmission at the same time.

The goal of Tor is to improve users' privacy by hiding routing information and communication content. However, Tor is not able to obscure the size, direction and timing of transmitted packets. Consequently, different peculiarities have been introduced in order to bypass side-channel attacks, such as: packets are fixed-size (512-byte) and known as *cells* (in this way attacks that fully exploit packets' sizes are bypassed), furthermore a specific defence has been introduced in Tor for misleading WF attacks [18], which consists in using HTTP pipelinig, aiming at randomizing the pipeline size as well as the order of requests. Despite this, Tor's defence is not successful in reducing the accuracy of such attacks [11]. For this reason, several new proposals and research in finding new features to counterattack these offensives have been advanced.

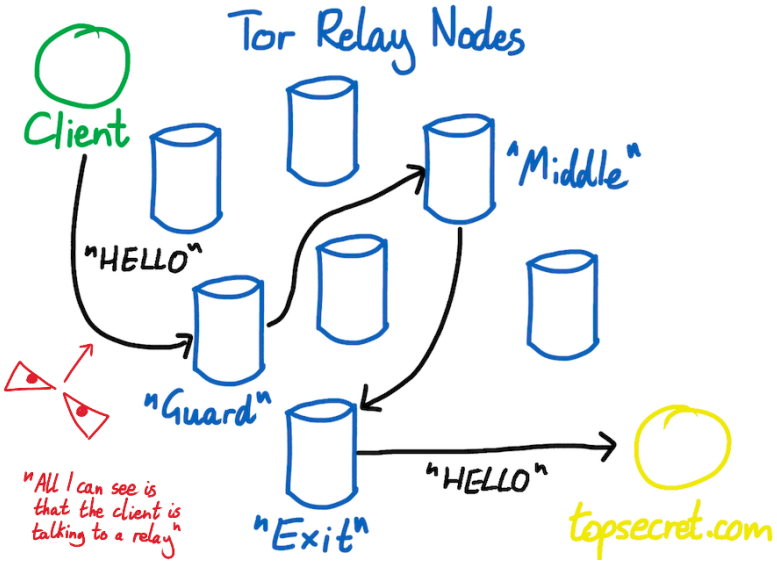


Figure 2.1: Tor network with local eavesdropper

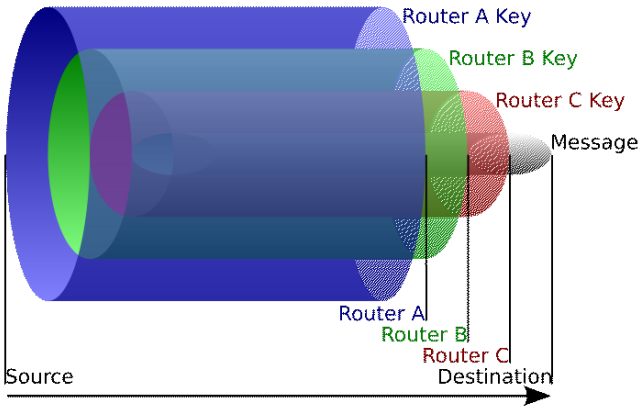


Figure 2.2: Layered Tor cell

2.3 Deep Learning and CNNs

"*Machine-learning technology powers many aspects of modern society*" [19]: this is how the founding father of convolutional networks [20], Yann LeCun, defines Machine Learning in a review published by Nature. Indeed, Machine Learning is a field of Artificial Intelligence which, by leveraging different models and mathematics, improved the state-of-art in various research areas, such as object detection and recognition, prediction of events, natural language processes, but also of dissimilar domains, such as genomics (like DNA sequence classification) or economics.

Deep Learning is a sub field of Machine Learning, and takes advantage of more complex and abstract models. One of the main advantages of DL over ML is that while the latter strongly depends on how the programmer extract the features, used for training the model, the former instead are good at capturing high- and low-level details, given the data. On the other hand, Deep Learning programs require a vast amount of data and burdensome computing power: that is why only recently it's attained so much usability, thanks to the growth of high-performance GPUs and development of big data.

Generally speaking, a deep learning model is inspired by the biological neural network of the human brain, so it is composed by layered, hierarchical neurons, which do some computations on their input and pass the result to the next layer. In figure 2.3 we can visualize an example of a dense model, where each neuron of the previous layer is directly connected to all the neurons of the following one (fully-connected), composed by up to 4 layers: the input layer, two hidden

layers and the output one.
Convolutional Neural Networks (CNNs) are specialized networks for processing grid-like structures, such as time-series data, images or videos, and are called in this way because they make use of a mathematical operation called *convolution*, in place of the general matrix multiplication, in at least one of their layers [21]: convolution allows the network to have sparse interactions (less computations and memory needed), shared parameters (reduce storage requirements) and to be equivariant to translation (thanks to the convolution operator and it means that if the input changes, the output changes in the same way; note that convolution is not equivariant to rotation nor scale changes).

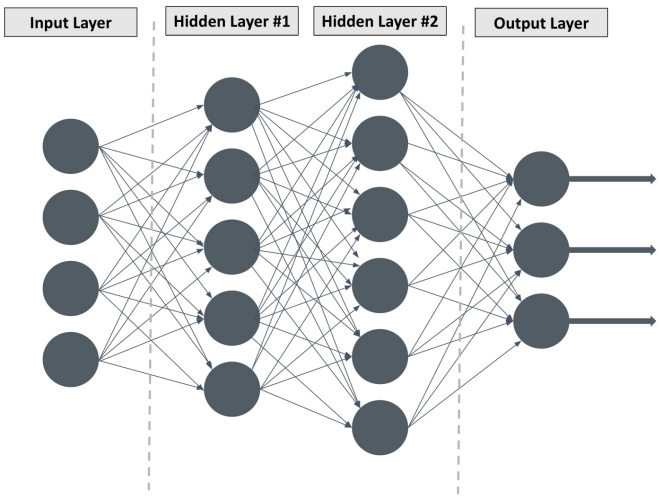


Figure 2.3: Example of fully-connected Neural Network

3
Related work

As WF attacks are gaining more and more relevance with the growing spread of privacy technologies and big-companies surveillance, a lot of researchers endeavour to hamper these threats against user's anonymity and privacy. In this chapter we are going to give an overview of some of the suggested defences through the years, in order to contextualize our work.
One of the most traditional and effective characteristics that distinguish WF defences is to delay packets, add padding and/or insert fake information. This cover traffic makes WF features less unique, thus decreasing the rate of classification accuracy performed by the malicious user. The very first defence to implement such strategies is the one of Dyer et al. [15], who conducted a review of low level defenses operating on individual packets. Dyer evaluates defenses using data released by Liberatore and Levine and Herrmann et al., who collected traffic from website homepages on a single machine with caching disabled. Their defence, named BuFLO (i.e., Buffered Fixed-Length Obfuscation), sends a constant stream of traffic at a fixed packet size for a pre-set minimum amount of time.

Another type of approach is the one proposed by Wright et al., who suggested the so-called *traffic morphing*[14]. This defence randomly pads packets, so that their lengths look as if they came from another distribution of packets, corresponding to another web page. This defence is able to bypass attacks that rely on packet lengths only as a feature and do not consider other features such as packet ordering. Indeed, later Wang et al. [8], exploiting the k-NN algorithm, proved that this defence was not valid.
Luo et al. proposed HTTPOS (HTTP Obfuscation) [22], which leverages features of the TCP

and HTTP protocols (particularly the HTTP *Range* header option) to affect packet size, object size, pipelining behavior, packet timing and other properties (such as the window size). Wang et al. [8] showed again that HTTPOS, as traffic morphing, has no effect on the final accuracy of the attacker.

In the same work in which Wang T. et al. propose their attack based on k-NN, the authors suggest a defence, which imposes to send a covering traffic sequence that is a supersequence[1] of the real packet sequence.

Thereafter, Wang T. et al. present another research [11], where they report a new bypass, named *Walkie-Talkie*, such that the client and the server employ a half-duplex way of communication (i.e., the client sends a request only after the server has fulfilled all previous requests and viceversa). In this way, the two peers send non-overlapping traffic of data. Moreover, they leverage burst molding, which consists of simulating the loading of two pages, by sending the supersequence of two burst sequences, which allows a much lower overhead than loading two pages consecutively. As we can observe in figure 3.1, the traffic flow of page 1 is paired to a decoy resource (page 2) and then, after the shift from full- to half-duplex communication, the supersequence between the sequences of the two pages is sent by the server to the client. Note that yellow squares stand for in-going cells, while the blue ones for out-going data.

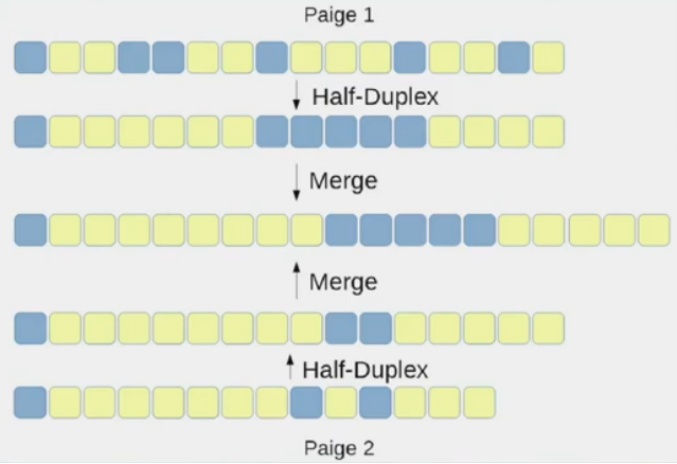


Figure 3.1: Idea Walkie-Talkie

Supersequence and Walkie-Talkie are two mimicry defenses which share the same usability issue: they require the client to have some information about web pages. Moreover, we believe

[1] in the Tor scenario, a packet flow can be considered as a sequence of -1's and 1's (as packets are cells); p is a supersequence of q if there exists a set of deletions of -1 and 1 in p to make them equal (maintaining order)

that Walkie-Talkie could be besieged by an inference attack: a malicious user can *infer* the requested resource, by analyzing multiple subsequent requests from the client, and deducing the information which is related to each other. For example, suppose a person has a singular disease; probably, she will look for symptoms (resource *A*), before, and for the remedies (resource *B*) then. Since in Walkie-Talkie to each real page is assigned a decoy one, suppose that resources A and B are assigned to pages C and D, respectively. The attacker, after the interception and examination of the network traffic, obtains as output of its attack that the client, in the first request, could have looked for resources A or C, while in the second request for pages B or D. Since A and B are correlated to each other, while C and D, very presumably, by construction, are not, the intruder can understand what the client is asking, compromising her privacy.

In 2015 Juarez M. et al. proposed *Website Traffic Fingerprinting Protection with Adaptive Defense* (WTF-PAD), a lightweight defence against WF attacks, which leverages adaptive padding. Adaptive padding is a padding strategy for low-latency that adds data only on those time interval in traffic that are probabilistically unlikely to happen. Together with W-T, WTF-PAD was considered the best defence which could be implemented in Tor. Indeed, WTF-PAD method can drop the accuracy of the k-NN attack from 92% to 17%, with 60% bandwidth overhead. W-T, instead reduces the accuracy of WF attacks to 50%, with 31% bandwidth overhead and 34% latency overhead due to the use of half-duplex communication. But in 2018, with the DL attack we mentioned before [10], Tor started to look for other proposes [23].

Recently, Gong J. and Wang T. [24] (2020) suggested two novel defences, namely FRONT and GLUE. The former works mainly during the so-called 'front-trace' , which is the first few seconds of a trace. Moreover, it adds dummy packets in a random way in both the directions, exploiting a Rayleigh Distribution. GLUE, instead, is a new way of defending from WF attacks, which is based on the 'split decision' problem. GLUE then, inserts fake packets between consecutive web page loads to obscure the distinction of the different loads.

In the same year, another protection against WF has been defined by De la Cadena W. [23]. Their defence, called 'TrafficSliver', consists in introducing randomness not in the traffic flow, but rather in the Tor routing algorithm. Their core idea is to distribute TCP traffic across different circuits built over several unique entries but shared middle and exit ORs (multipathing approach).

It is worth mentioning the work by Nithyanad R. [25] (2014), based on SSH traffic, in which they exploit clustering in order to generate a matrix that can convert a traffic flow to another one. We will talk about it in chapter 7. In table 3.1 we sum up all the WF defences described in this chapter, with a short description and the relative limitations.

WF Defence	Summary	Limitations
BuFLO [15]	it modifies the traffic to make it look constant rate	bandwidth and time overhead
Decoy [13]	it loads a randomly chosen web page (background page) simultaneously with the actually requested one (camouflage)	bandwidth and time overhead
Supersequence [8]	it computes supersequences over sets of packet sequences	bandwidth overhead & large database of web page templates
Glove [25]	it generates offline a transformation matrix which will be used to generate covert traffic	bandwidth overhead & large database of web page templates
Walkie-Talkie [11]	it employs a half-duplex communication and burst molding (i.e., supersequence between two web pages)	a priori knowledge, inference attack & memory overhead
traffic morphing [14]	it maps packet sizes from one site to a packet distribution drawn from another site	no effect on the accuracy of more recent attacks [8]
HTTPOS [22]	it exploits TCP and HTTP features such as HTTP range header	no effect on the accuracy of more recent attacks [8]
FRONT [24]	it creates random noise at the beginning of a page load both from the client and the server	accuracy of more recent attacks remains high [23]
GLUE [24]	it adds dummy packets between consecutive page loads to obscure separation of different page loads	accuracy of more recent attacks remains high [23]
WTF-PAD [26]	it uses adaptive padding technique	no effect on the accuracy of more recent attacks [10]
TrafficSliver [23]	it introduces randomness into the Tor routing algorithm (multipathing)	no protection against local eavesdroppers
Divergent	it exploits randomness, binomial distribution and dummy packets	server resources analysis

Table 3.1: WF defences.

4

Identifying resources using Neural Networks

In this chapter, we present how the attacker model is defined and which features are exploited both in the traditional network and in Tor. Then, we deal with how we collected the data for our WF attack and the way we employed it in order to extract the useful features to train the offensive model. Finally, we explain the experiment in the anonymize setting and the difference between the two frameworks.

4.1 Adversarial CNN-model

The CNN model we conceived for setting up the attack we discussed in section 2.1 is inspired by the ones proposed in the state-of-the-art [27, 6, 10]. It is composed by:

- one input layer
- five convolutional layers;
- four dropout layers;
- one max-pooling layer;
- two dense layers
- one output layer.

The activation functions exploited are: all but the first layer exploits *ReLU*, the first hidden layer uses the Hyperbolic Tangent function (*tanh*) and the output layer *softmax*. These have been chosen both empirically, but also because: ReLU, which is a linear function for positive inputs, while it outputs 0 for negative input values, overcomes the vanishing gradient problem (introduced by tanh), allowing models to learn faster and perform better and is the default activation when developing convolutional neural networks; tanh introduces non-linearity and, finally, softmax is widely applied for multi-class classification problems.

In figure 4.1 the model is plotted. Note that the final layer outputs as many values as the number of total classes (*num_classes*) of the multi-class classification problem. This characteristic comes from the softmax activation function. Specifically if the network is configured to output a vector of N values, where N is the number of classes in the classification task, the softmax function is used to normalize the outputs, converting them from weighted sum values into probabilities that sum to one, so each element in the output-vector of the softmax function is interpreted as the probability of membership for each class. The softmax function is defined as following:
$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\, i = 1, 2, \ldots, K,$
and takes as input a vector ($\vec{z}$) and normalizes it into a probability distribution, which consists of the same dimension as the input. This output vector (also called *confidence*) will be useful for our evaluation discussion in chapter 6.

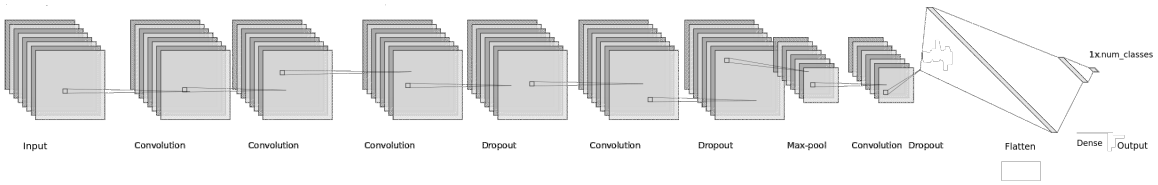


Figure 4.1: Adversarial Convolutional Neural Network architecture

The main features for our adversarial DL model, in general, are the following ones:

- global size (i.e., summation over each packet's size);
- size of each packet;
- inter-arrival time of two consecutive packets;

- packet density (i.e., how many packets are received within a time interval)

The way these features have been extracted and why these characteristics can be important for the attacker are described in the following section 4.2.

Before concluding this section, we would like to clarify to the reader why a CNN is one of the most efficient model to leverage for such attacks, giving an example: CNN are the state-of-the-art deep learning infrastructure for dealing with, among all, images (which have a grid-like structure). Images are nothing more than a matrix of pixels, whose values are different depending on the pixels' color. A packet sequence can be interpreted, indeed, as an image, in which each row contains the features' value, and if the features' arrays are different in dimension, pad them with worthless quantities.

4.2 Data collection & features exploited

In the scenario with traditional network, we implemented a setting in which a client can request 10 different videos to a server and the server responds with the corresponding resources. In the meanwhile, an attacker, who is in the same LAN of the client, intercepts the traffic of packets coming from the client to the server and vice versa. Since the channel of communication built between the client and the server is encrypted, the adversarial is not able to decrypt and get acquainted of the payload inside the packets: so he relies on side-channel features, which are present in the communication packets by construction. Side-channel attacks which are only based on the general concept of *size* can be bypassed in different and efficient ways [27, 11]. So, in order to implement an adversarial model, which is able to take advantage of the peculiar characteristics which compose the traffic flow, at least other features need to be taken into account. Especially, as we can see in table 4.1, for each session we considered: the **total amount of packets exchanged** (in byte), the **size of each packet**, the **inter-arrival time** between two packets and finally the **packet density**, which represents the quantity of information exchanged during a certain period of time.

The first two features are based on the dimensions of the transferred data. These can be helpful, above all, in scenarios in which no countermeasures against WF attacks are considered, or when a defense is implemented, but this does not completely remove the size as a helpful feature for the attack to be executed (note that the majority of the suggested proposals does not completely remove the benefit of this feature since it requires a lot of overhead in order to completely bypass it [14]). The third and the fourth features are useful for the adversarial CNN model in

order to consider also the network architecture's specifics, which compose the devices between the client and the server.

Feature	Description
Total size	It counts the total number of exchanged bytes during the connection
Packet size	It counts the byte-size of each packet
Inter-arrival time	It represents the time interval that is between two consecutive packets
Packet density	It counts the number of packets that are transmitted during a fixed time interval

Table 4.1: Exploited features in the traditional network setting and relative description.

Feature	Description
Packet direction	It defines if the packet is incoming or outgoing
Packet arrival-time	It represents the arrival time of each packet

Table 4.2: Exploited features in the Tor network setting and relative description.

As regards the Tor setting, we take advantage of the dataset[1] made available by Wang T. (2021). In Tor we recall that each packet, named cell, is fixed-size and prior work has shown that, for this reason, we can consider this type of traffic flow as a sequence of -1's and 1's (where the sign represents the direction of the data), without losing accuracy in the attack [10]. Note that, in the previous scenario, the values of the sizes of real packets help to improve the efficacy of the villain's aim. Besides this difference, since the dataset is composed by the arrival-time of each cell, rather than inter-arrival times, we opted for the former. Furthermore, in this setting we take into account the direction of the packets, too. In table 4.2 we represent them.

[1] The dataset for the Tor setting is reachable through the following link: https://www.cs.sfu.ca/~taowang/wf/

5 Our proposal - Divergent

Our proposal's aim is to reduce the amount of time and bandwidth overhead (treated in depth in 5.3) introduced by the defence, keeping a high level of effectiveness. Moreover, some state-of-the-arts introduce spikes on the traffic flow [12] [11]. This could be a problem, since if we think that a potential user is surfing more websites contemporaneously, and we hypothesize that each of these sites/servers implement such measures, then the device of the client could be flooded by these network traffic's spikes (for example, imagine a student who is watching a video-lecture and at the same time playing LoL).

In this chapter we define the attacker model, both in traditional and anonymized networks. Then, we specifically describe how Divergent works and its main characteristics. Finally, we illustrate the impact of randomness in the general world of cybersecurity and, specifically, of our work.

5.1 Threat model

In this work we examine two distinguished threat scenarios: the first one considers a client and a server communicating through a traditional network. In this setting, our threat model considers a client-victim that requests resources to a web-server, over an encrypted connection (e.g., exploiting the TLS protocol).
We assume that the adversary is *passive* and *local* to the client, so he can directly observe the victim's traffic and knows the client's identity, as well as he is hard to detect, since he never tries

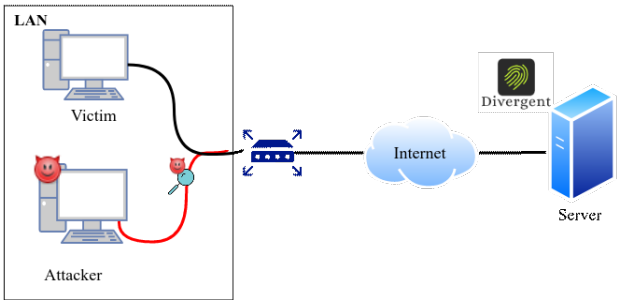
to modify the client's packet sequence or inject packets, as a passive eavesdropper. In order to do so, he has complete access to the device he exploits, so can gain, among all, root privileges (needed for capturing raw packets with whichever tool, such as tcpdump [28] or Wireshark [29]). The adversary cannot break standard cryptography (e.g., if the client-victim and the server employ the TLS protocol in order to communicate, the attacker cannot read the plaintext, unless he knows the necessary secrets), impersonate nor compromise the target. Finally, the attacker's aim is to identify the resource the victim is asking for.
An adversary who is able to determine the resources the users is requesting is a significant privacy threat, which could be exploited, for example, by an attacker in order to steal sensitive information, such as the disease a user has, or by a government for mass surveillance.

The second scenario is similar to the previous one, except that the client surfs the net using anonymization technologies (particularly, in our work we focus on Tor and all its implemented features), that is, the client connects to the Internet through one or more proxies.

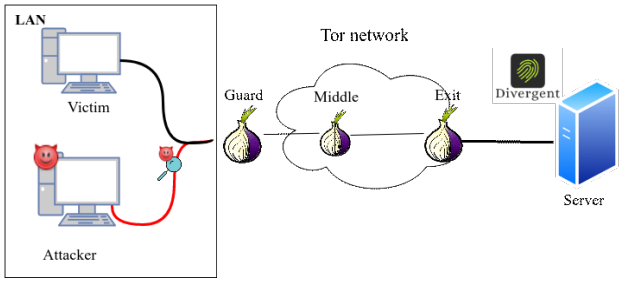
The current design of Tor's network assumes the absence of a global adversary that is able to monitor both ends of the communication, i.e., entry, and exit guards, denying in this way anonymity. Indeed, in the analysis of the security of the onion router provided by Syverson, et al. [30], the probability that an adversary owns either the first or the last node on the circuit is the same and equal to : (the number of compromised nodes (c)) / (the total active nodes in the system (n)). Therefore, an adversary that is able to compromise the first and the last nodes on the route might exist with probability = c^2 / n^2. In addition to this, these probabilities are only valid for a certain interval of time, as routes are dynamic, which means that after a specific amount of time, it will be automatically torn down (except for the guard node) and another path will be established for future activities.

Consequently, in our model the attacker can, at most, compromise the guard node of the victim. As the attacker is local to the client, he knows the victim's identity but not the server she is talking to, nor the web page (or the resource) she is visiting. Note that we assume at least one proxy cannot be threaten by the attacker, otherwise previous work showed that if the adversary controls both the end-points of a multi-proxied connection than he can compromise the entire privacy of the victim [30]. Before the client could connect to the network and send requests, she must goes through a preliminary phase and create the circuit of relays, as we have already mentioned in section 2.2. The non-endangered nodes' aim is to protect the victim and her privacy.

In figure 5.1 we represent the threat models in the two distinct scenarios.



(a) System model scenario in a traditional network



(b) System model scenario in a Tor network

Figure 5.1: System models in the two different scenarios

In general, the attacker's strategy is the following one: the attacker collects packet traces from several web pages that he is interested in monitoring. Then, he observes packet traces generated by the victim during her web browsing, and compares these traces with the ones he collected by performing supervised classification.

For the sake of completeness, observe that the majority of normal-users still don't use very frequently, in general, technologies like proxies or Tor in order to accomplish their regular tasks; indeed, less than 1/3 of users all over the world employ VPNs every day [31] and roughly 2 million users access the Tor platform daily [17] and considering that almost 5 billion people use the internet every day [32], less than 0.05% of people surf the net exploiting Tor day-to-day.

5.2 Our countermeasure: Divergent

The solution we propose is the same for the two different types of networks (i.e., traditional and anonymizing), so we are going to describe it with the traditional setting as background.

The server, after having created a secure channel with the client, sends to her a secret, random number x. So, supposing that both the peers agree on the TLS protocol and share the necessary secrets and information (e.g., cipher specifics, certificates, session keys, ...), immediately after the last TLS message, the server transmits, encrypted, x .The value x represents the number of packets that the server has to send to the client, before delivering a burst of dummy packets. Moreover, since we noticed that transmitting only one dummy after x real packets does not decrease the attacker's performances, we opted to send 1 or more dummies per burst. The number of dummies sent in the burst is equal to y, it is randomly generated and next we explain why the server does not need to deliver this parameter to the client. The idea of inserting such dummy packets is to create a different traffic flow, w.r.t. the original one, in order to disrupt the adversarial model, so that it is less capable of recognising a stream associated to a resource from another one. Furthermore, since x and y are randomly generated, the packet-stream changes each time a new request is performed.

The quantities assigned to x first and y, during the communication, are fundamental for a twofold reason: the first one is the overhead stuck in, while the latter is the effectiveness of the defence. With regard to the overhead introduced, if y is too big (that is, if a lot of dummies are sent per burst) or x is too small (i.e., the server sends fake packets after a small number of real packets), then a lot of futile data is introduced. On the other hand, suppose that the server contains only two resources (A and B) that can be sent to users, and A is way bigger in size than the other one. Then, the traffic fingerprinting associated to A is very dissimilar to the one of B (if we think, for example, on the number of packets transmitted, the time that it takes to send it whole, etc.). So, the values x and y's, adjusted for the resource B, must be smaller and bigger, respectively, than the ones associated to A. In this way the attacker's model will have more difficulties in distinguishing the data, as the traffic shape of B resembles more the flow of a bigger resource, that is the A's one, and vice versa.

We sample x and y, as we reported in the pseudocode of Divergent of Algorithm 5.1, exploiting the *binomial distribution*[1]. Specifically, a resource which is bigger will be assigned to a Probability Mass Function (PMF) which, probabilistically speaking, will add less packets, with respect to a resource which is smaller. To do this, we leveraged the n parameter of the binomial distribution, whose value varies among 36 at most, and 3 at least. We decided to choose the binomial distribution in order to obtain the values of x and y, because there is possibility that

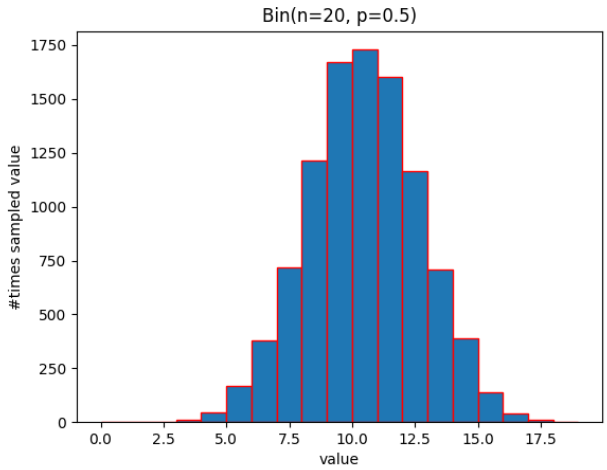
[1] Remark that a binomial distribution $Bin(n, p)$ is a discrete probability distribution, defined by two parameters, n and p, which set the number of times the experiment is repeated and the probability of success for each time, respectively.

the network fingerprinting associated to a small resource appears to be the one of very big data, but at the same time of medium-size resource, too, prioritizing values that do not introduce too much expenses. Indeed, as we can observe in figure 5.2, by construction of the binomial mass function, the numbers sampled in the distribution on the top 5.2a (where $n = 20$) are, with more probability, bigger than the ones of the distribution below 5.2b ($n = 12$), but it could happen (less frequently) that this does not hold. Observe that the more we approach the values near the mean of the binomial distribution (where its mean is equal to $n * p$), the more they are likely to be sampled.

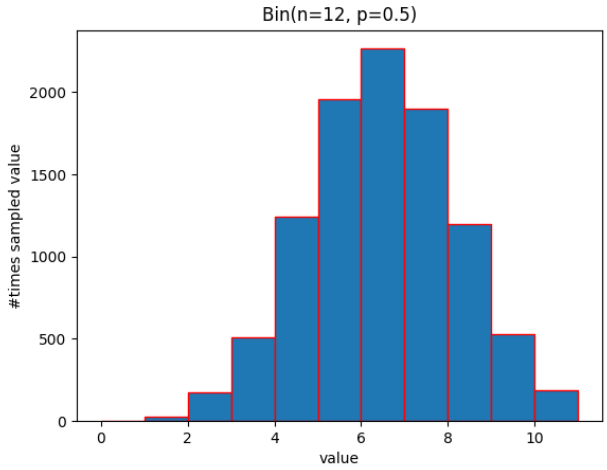
Note that, while x remains fixed throughout the whole transmission of the resource, the burst's size (i.e., y's value) can change for each burst of dummies (next the reason why will become more clear). To clarify, in figure 5.3 on the right, we can observe that the server, after sharing the needed data for building a TLS channel with the client, sends the secret number x. In this specific setting, x is equal to 2. Then, at first, the client receives one dummy packet after getting $x = 2$ 'real' packets; then, she receives two dummies, after obtaining packets 3 and 4. So we can see that y is equal to 1 in the first burst, and to 2 then.

The receiver, in order to recognize if she got a dummy packet or real data, can rely on the x value. However, it is commonly known that, in whichever network, a packet not always is able to reach the other counterpart. So, in order to bypass this problem, when the server has to send fake data, it inserts the '*dummy*' writing at the beginning of the payload. The x value then, comes in handy for adding redundancy to the transmission protocol.

As we mentioned before, the quantity x, once sampled, remains constant throughout the whole TCP connection, while the number of dummy data sent per burst can change. The reason why x remains invariable is not immediate, so we start giving an example: suppose that the server has to send the smallest resource r_1 to a client. We remark that the aim of Divergent is to create a traffic flow which is not unique to each resource, but that puzzle the adversarial model. Since r_1 is the smallest, as we observed before, the server will sample a little x, and depending on its value, the cover traffic associated to r_1 could be similar to the one of the second smallest resource r_2 in the server, or of a bigger one. If, instead, the x values changed, the cover traffic could be associated to both a mixture of the traffic flow of r_2 and of bigger ones, which can create another unique fingerprint of r_1, leading to a not efficient defence. Therefore, x is invariable for: lowering the computation of the server (since the server does not need to continuously sample x), not introducing too many perturbations in the transmission, and, if needed, for removing "dummy" from the payload of the dummy packets.



(a) Binomial distribution with parameters $n = 20$ and $p = 0.5$



(b) Binomial distribution with parameters $n = 12$ and $p = 0.5$

Figure 5.2: Plot of two binomial distributions

For what concerns y, we defined it so that it can change value per burst of dummy packets. Indeed, y changes because if it remained constant during the whole TCP connection, the server would send burst of dummies of same size (e.g., if $y = 3$, then each burst is composed by 3 dummies), which could be noted by the attacker, unless the defence didn't leverage the timing of sending each dummy packets, but this would introduce time delay. Therefore, the size of each spike of dummy packets changes mostly to disrupt the adversarial, since the traffic fingerprint changes continuously and with more preponderance.

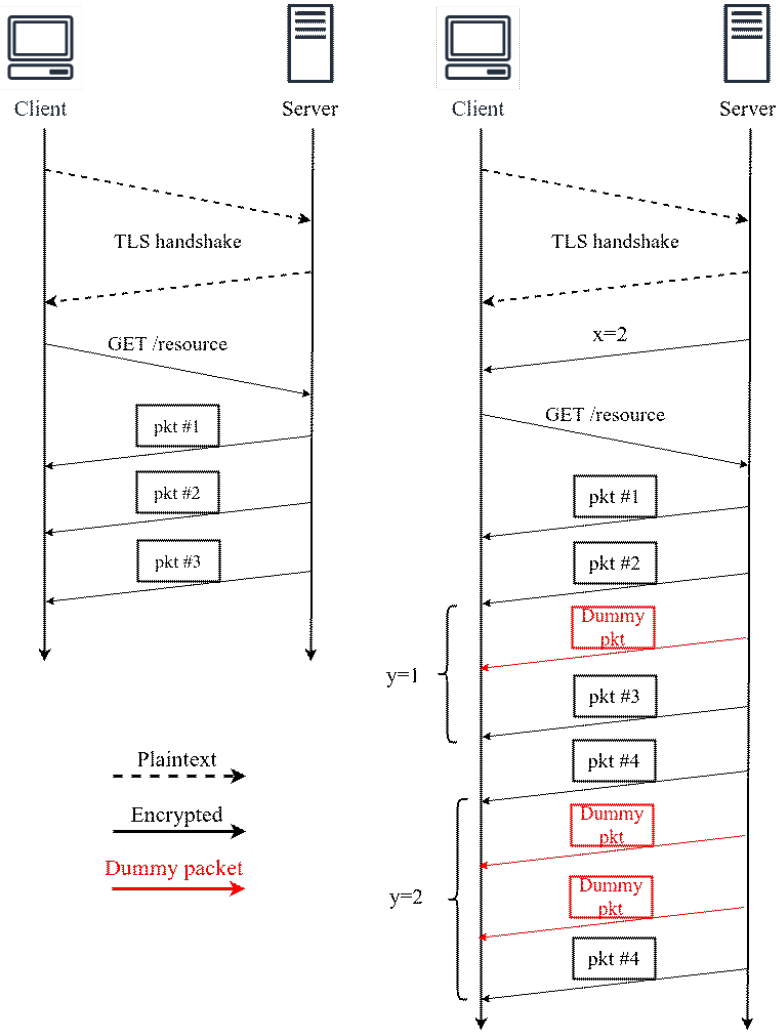
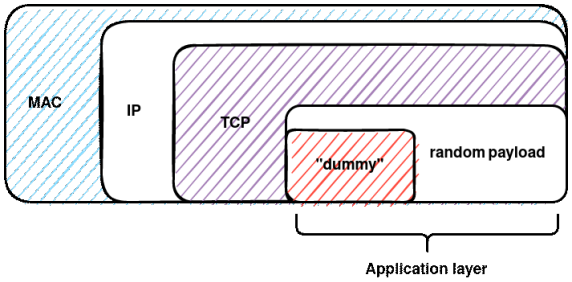
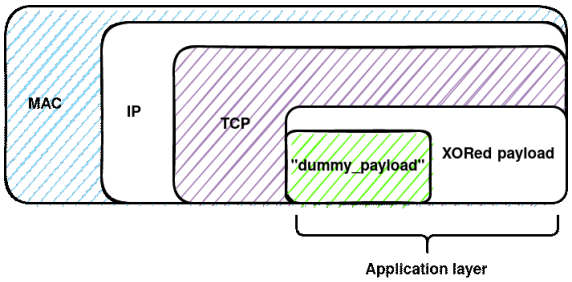


Figure 5.3: Example scenarios without (left) and with (right) Divergent

In order to decrease the bandwidth overhead, we decided to insert, in some dummies' payload, the output of the *xor* operation between the last packet received before the dummy packets with a following one. So, considering figure 5.3, the payload of the dummy packet in the first burst could contain (see section 5.3) the output of the *xor* operator of the 2*nd* packet's payload with the payload of the 3*rd* one. The selection of the packets which contain helpful data is per-burst (i.e., if a dummy packet does not have useful data in the payload, all the dummies of the burst won't have it, and vice versa) and there is 50% of probability that the immediately next burst of packets contains the payload of the following information which needs



(a) Dummy with Random payload



(b) Dummy with XORed payload

Figure 5.4: Construction of dummy packet with random (top) and xored (bottom) payload

to be sent. One may think: "why don't the server introduce xored payload in all the dummy packets?". The answer is plain: to introduce another level of uncertainty in the transmission of the resource. Another legitimate doubt could be: "why do you insert a *xored* payload, and not directly the needed data?". The response is that by xoring packets, we introduce another small level of uncertainty to the attacker, given by the fact that, if the two xored packets are different in size, than we need another dummy for containing the remaining result. But this does not happen always, only when the following, real, packet is bigger than the real previous one.
The server, for the purpose of making the client recognising which dummies holds useful data or not, appends to the dummy's data section the "*dummy_payload*" writing.
In figure 5.4 we represented the construction of the two different typologies of dummy packets. To sum up, once a client receives a packet: if the packet's payload starts with "dummy" (as represented in figure 5.4a), then she knows she received a packet with useless data, so discards it; if she gets a packet, whose data section begins with "dummy_payload" (as in figure 5.4b), then the packet is kept; otherwise the client received a *normal* packet.

With this defense, by construction, no spikes are introduced in the communication between two peers, and less overhead is introduced, thanks to the information eventually carried by the

packets in the burst.
The pseudocode of algorithm 5.1 specifies how practically, and technically, Divergent works: from line $1 - 6$ the server samples x and then sends it to the client. Next, the server defines an index (i.e., *i_random_dum*), which counts when it is necessary to send the burst of packets, and starts to read the the first byte of data. The while-loop of line 9 continues until there is data that should be sent. If the server enters in the first if-statement of line 11, it means that it is sending real data, otherwise it enters in the else-body of line 16 (i.e., the index for sending dummies is equal to x). Inside the else, the server computes the size of the burst (i.e., *num_burst*) and if the burst of packets will be useless or not (i.e., the boolean *has_payload_dummy*). Inside the for-loop of line 19, the server sends dummy packets with *xored* or random payload, based on the value of *has_payload_dummy*. After having sent all the dummy packets, resets *i_random_dum*'s quantity and starts from the beginning of the while-loop.

5.3 Bandwidth Overhead and Time Overhead

The main ideas of proposing a defence are (1) bypassing the considered attack(s) and (2) impact as less as possible the *efficiency* of the infrastructure we are trying to protect. Two parameters that are fundamental in order to understand if a proposal can be a good one, are *bandwidth* and *time* overheads.
Bandwidth overhead can be defined as the number of dummy byte inserted in the traffic flow by the defence, divided by the transmitted bytes of the same resource without the defence, times 100.
Time overhead, instead, is the extra amount of time required to transmit a requested data, divided by the original amount of time of the same information sent. It is important to notice that a large time overhead worsen the client's experience, as the client needs to wait longer to load web pages.
In our work, the time overhead is increased by the following features: sampling and sending x, sampling y and the *xoring* operation. Moreover, the delay overhead depends on: if the burst of dummy packets does not contain a useful payload and the application data, in the server, is not ready, than there is no added delay, otherwise the delay increases of the needed time for sending the burst of cover packets (transmission time). On the other hand, if the burst is with the xor of the real payload, then it halves time delay introduced by dummies without payload and with data ready to be sent. In table 5.1 we sum up the four different cases and if the table cell contains "X", then no time delay is inserted, vice versa with the "✓" symbol. The scenario in which the

Algorithm 5.1 Pseudocode Divergent

Require: client c and server s established an encrypted channel.

▷ Computation of x.
$x \leftarrow 0$
while $x = 0$
    $x \leftarrow Bin(n, p = 0.5)$ {n depends on r's size.}
end while
s sends x to c
$i\_random\_dum \leftarrow 0$ {used to count when sending burst of dummy pkts.}
$data \leftarrow read\_file(r, MTU)$
while $data$
    ▷ Enter in the if-body if we have not reached x.
    if $i\_random\_dum < x$
        $i\_random\_dum \leftarrow i\_random\_dum + 1$
        s sends $data$ to c
        $data \leftarrow read\_file(r, MTU)$
        ▷ Else, send burst of dummies with/out payload.
    else
        $num\_burst \leftarrow Bin(n, p = 0.5)$ {nr. of consecutive dummies to send. It must be > 0. n depends on r.}
        $has\_payload\_dummy \leftarrow Bin(n = 1, p = 0.5)$ {Bernoulli trial.}
        for $i = 0$ to num_burst
            $len\_dummy \leftarrow Bin(n = MTU * 2, p = 0.5)$ {length of dummy packet. Must be $\leq MTU$.}
            ▷ If enter in the if-body, send dummy with payload, which is the xor of the last 'real' packet sent, with the following one. Otherwise, send dummy with junk payload.
            if $has\_payload\_dummy$
                $next\_data \leftarrow read\_file(r, len\_dummy)$
                $xor\_payload \leftarrow data \oplus next\_data$
                $dummy\_pkt \leftarrow$ "dummy_payload" $+xor\_payload$
                s sends $dummy\_pkt$ to c
            else
                ▷ Generate random payload with length len_dummy to append to dummy packet.
                $dummy\_pkt \leftarrow$ "dummy" $+random\_pay$
                s sends $dummy\_pkt$ to c
            end if
            $i \leftarrow i + 1$
        end for
        $i\_random\_dum \leftarrow 0$
    end if
    $i\_random\_dum \leftarrow 0$
    $data \leftarrow read\_file(r, MTU)$
end while

application data is not ready and the next bust of dummies is obviously not possible.

	Dummy **w/ payload**	Dummy **w/out payload**
App data **ready**	X	✓
App data **not ready**	not possible	X

Table 5.1: Different scenarios for time overhead

Employing our PC and Python as programming language (since our project is mainly based on it), numerically speaking we obtained the following timing (on average):

- the time for sampling a number from a binomial PMF is $3.1us$;
- the time for performing a xor operation is $0.357us$;
- finally, the overall time bandwidth, as we mentioned before, depends also on the transmission time. Supposing we are in a very poor scenario, where the MTU is equal to $512Byte$ (cell size in Tor) and bitrate is less than $1.5Mb/s$, then we obtained a final time overhead of 0.65%, as result of the two just computed timings, summed to the transmission time for sending packets.

To make things more clear, suppose that: the number of dummy packets which contain payload is defined as d_{pay}, the number of bursts of dummies in the whole transmission is n_{burst}, the number of dummy packets without payload is d_{fake}, the original amount of time to send the data without defence is $t_{original}$ and the timings for *xoring*, sampling and transmitting are respectively t_{xor}, t_{sampl} and t_{tx}. Then, the mathematical expression we employed to compute such overhead is the following one:

$$time\_overhead = \frac{(t_{original} + d_{pay} * t_{xor} + n_{burst} * t_{sampl} + d_{fake} * t_{tx}) - t_{original}}{t_{original}} * 100 \quad (5.1)$$

$$= \frac{d_{pay} * t_{xor} + n_{burst} * t_{sampl} + d_{fake} * t_{tx}}{t_{original}} * 100 \quad (5.2)$$

$$\approx \frac{d_{fake} * t_{tx}}{t_{original}} * 100 \quad (5.3)$$

Note that the amount of delay inserted by Divergent is very low, since: the *xor* operation and sampling times are negligible (even if we took into account their values in the computation of the overhead). That is why *time_overhead* can be approximated as we can see in equation 5.3.

As well, the transmission times for sending dummies without payload is, in general, quite irrelevant with respect to all the packets in the transmission.

As regards the bandwidth overhead, Divergent leverages packets' distribution and size: the idea of injecting dummy packets with *xored* payload (with 50% of probability per burst) allowed us to reduce the initial overhead from 50% to 23% on average. In table 5.2 we reported the time and the bandwidth overhead of the main defences. Considering that the work of Wang T. et al. [11] (Walkie-Talkie) was one of the candidate defence to be implemented in Tor [10] has 31% of bandwidth overhead, Divergent improves the efficiency of one of the main proposals.

WF Defence	**BWOH**	**TOH**
BuFLO [15]	145%	180%
Decoy [13]	100%	39%
Supersequence [8]	222%	112%
WT-PAD [26]	64%	0%
W-T [11]	31%	34%
Divergent	23%	0.65%

Table 5.2: Bandwidth (BWOH) and Time (TOH) Overhead of some of the WF defences.

5.4 Importance of randomness

As we have seen throughout our proposal, Divergent heavily depends on the generation of random numbers, such as the secret value x, which defines how often dummy packets must be sent, or *has_payload_dummy*, defined in line 18 of 5.1, which is a boolean variable and determines if the next burst of dummies will contain junk data as payload or the xor of real payload. If an attacker was able to understand the values assigned to these variables, he could re-construct the traffic flow without the defence, use this reconstruction as input-test to his adversarial model and get the resource requested by the victim with way higher confidence. For this reason, we introduce some of the main concepts about PRNGs and clarify how relevant randomness is.

First and foremost, random numbers are typically sourced by two kinds of random number generators: True Random Number Generators (TRNGs) and Pseudo-Random Number Generators (PRNGs). Generally speaking, the former exploit as entropy sources physical phenomena (such as thermal noise or photoelectric effect), while the latter leverage deterministic algorithms and, in computers, use the attached hardware to harvest entropy like movement on the mouse pointer, keys typed on the keyboard, and disk and/or network I/O. However, true RNGs on their own are often expensive and can be subject to gradual decline over time. For these reasons, most of mechanisms and infrastructures rely on a PRNG.

A PRNG must be secure against both external and internal attacks. The attacker is assumed to be aware of the code of the generator, and might know the entropy used for refreshing the generator's state. Now we introduce the basic security requirements, which a generator should satisfy [33]:

- *Pseudorandomness*: the generator's output looks random (independent and identically distributed values, i.e., i.i.d.) to an outside observer;
- *Forward security*: an adversary who learns the internal state of the generator at a specific time cannot learn anything about previous outputs of the generator;
- *Backward security*: an adversary who learns the state of the generator at a specific time does not learn anything about future outputs of the generator, provided that sufficient entropy is used to refresh the generator's state.

In our work, we ran our experiments in an OS based on the Linux kernel. This kernel presents an its own, software-based, pseudo-random number generator (denoted ad LRNG). In Linux, everything is considered as a file, and the device files */dev/random* and */dev/urandom* are the main interfaces to the crypto PRNG which can reliably generate random bits. The kernel maintains an entropy pool which is used to store random data generated from events like inter-keypress timings, inter-interrupt timings, etc. etc. The main differences between the two special files is that /dev/random is typically blocked if the amount of available entropy is less than the requested. On the contrary, the /dev/urandom is never blocked, even if the PRNG seed is not fully initialized with entropy since boot. Note that, starting from 2020, the Linux kernel 5.6 /dev/random only blocks when the PRNG hasn't been initialized. Once initialized, the /dev/random and /dev/urandom behave the same [34] [35].

In Divergent, it's particularly important that the values returned from the Binomial distributions are as more i.i.d. as possible. Remark that a Binomial of parameters n and p outputs the

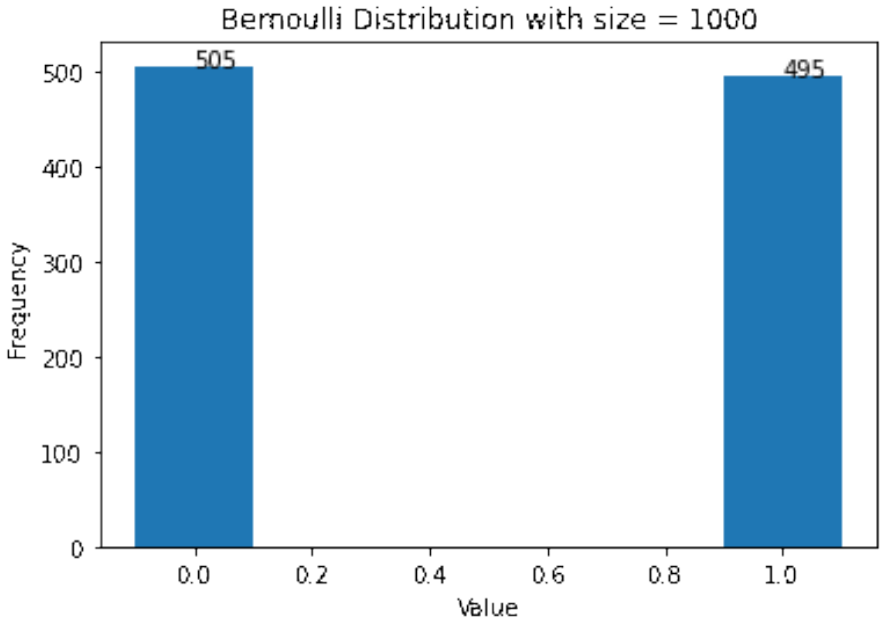


Figure 5.5: Bernoulli distribution implemented in our PC

number of successes in n independent Bernoulli trials. To this aim, we tested that our Bernoulli function gives as output the losses, 0, and the successes, 1, equal in number. In figure 5.5, we can observe a result of this experiment, run 1000 times and we observe that it approximates a uniform distribution. Notice that the more we increment the size, the tighter it is to a uniform distribution function, i.e., asymptotically tends to be uniform.

6 Performance evaluation

In this chapter we evaluate the performances obtained by the attack we described in section 5.1, against traffic data without Divergent and with the defence in place, in both the settings in which client and server employ a traditional network or the Tor infrastructure.

6.1 Evaluation setup

For our experiment we needed to implement both the attack and the defence. The attacker model, which is a CNN, has been built using Python, in particular with the TensorFlow library [36], coupled with Keras [37] as a front-end interface. The reasons why we chose TensorFlow as the framework are multiple, such as: simplicity, well known by the research community and if paired with the Google Colab environment [38] it allows free access to Google Cloud's GPUs, speeding up the training and test processes.
The data we used for running our experiment is composed by 10 different videos, which have been made available by J. Lokoc, B. Münzer, K. Schoeffmann, M. Del Fabro, M. J. Primus, T. Skopal, and J. Lánský [39].
We collected the data for the attacker in the traditional setting by implementing a web server and a web client with Python. Both peers communicate through the *localhost* interface. While the victim and the server send data to each other, the malicious user intercepts the traffic packets using tcpdump. For each video, we created 100 instances, so we ended up with 1000 data inside the dataset. The dataset, in order to train and test the offensive model, has been split in the

following way: the training set is composed by 60% of the whole data, 30% composes the validation set and the remaining data is the test set[1].

As regards the hardware setup we used an ASUS laptop, the VivoBook S, characterised by and Intel Core I7-8550U with base frequency 1.80GHz. The GPU is a GeForce MX130 one, with 4 GB.

6.2 Evaluation metrics

We assume that the user is limited to request a fixed set of videos, and the attacker knows this set and can train his classifier on it. The success of the attacker is measured as the ratio of the number of correctly classified traces to the total number of traces ($P_{correct}/N$). This ratio is also named as the attack's *accuracy*. $P_{correct}$ is the total number of correct predictions, while N is the total number of instances in the test set. A correct prediction is defined as the output of the classifier, which matches the label of the resource to which the test trace belongs.

6.3 Results

In the setting where a traditional network is employed, we use 10 distinct videos as data. The accuracy obtained by the malicious user with traffic without any defence against WF attacks is equal to 92.86%, while with Divergent it decreases to 49%. It can be seen still as a good accuracy obtained, but the main advantage of the defence is that the adversarial model's confidence is very low. This means that, even if the output is correct, the model has a lot of "indecision" on it. Remark that a machine learning model, in general, for a multi-class classification problem, assigns to each input a class with a confidence value and, finally, it outputs the label with the highest certainty (the summation of all the confidences, of each input, must be equal to 1). As we can see on table 6.1, where we reported the results of the arrays of confidences obtained by the attacker for true positives outputs only, all the confidences are less than 50%, and even worse for videos 2 and 5, where the confidence is less than 16%. Observe instead how the CNN is uncertain between videos 7 and 4 in the 3*rd* row of the table: 49.8% against 44.3%. These results directly explain that, even if the attacker model makes a good prediction, it is not sure about its output, and this is exactly what a WF defence should behave: insert uncertainty in certainty.

[1] We leave the link to our code and data: https://github.com/francevarotz98/Divergent

For what concerns the Tor settings, we remark that we employed another type of data (by Wang T. et al.). In this setting, the final score of the adversarial user without protection is 95%, while it drops off to 29% with the defence. We note that our defence works better in the Tor scenario. We deem it's due to the fact that the data in the previous setting has been taken in a *localhost* scenario, so the (inter-)arrival times are very tiny. In this way, even if we add dummy packets, the DL model is not as obstructed as for the data of Wang T., because the attacker is able to detect and distinguish the dummy packets, due to the fact that sampling y's values takes one order of magnitude than without such computation (as we have seen in section 5.3).

Figure 6.1 represents the confusion matrix of the results of the Tor setting we just mentioned, and shows the difference of performances of the attacker model between Tor data without Divergent (on the left) and Tor data with Divergent applied (on the right). A confusion matrix is a visual tool, whose usage is to portray the quality of the output of a classifier. It works as follows: the x-axis represents the labels that the model has assigned to the input (*Predicted label*), while the y-axis stands for the actual input's class (*True label*). Therefore, the higher the values on the diagonal of the matrix, the better the classifier works. As we can see from the figure, the matrix on the left is almost diagonal[2]. Indeed, as we obtained only 5% of mis-classification without the defence, only few videos have been not correctly classified: for example, video 0 has been wrongly predicted once as video 1 and once as video 4, while the adversarial classifier always recognizes video 9.

The matrix on the right, instead, has a lot of scattered values. This means that, thanks to Divergent, a lot of data are mis-classified by the adversarial model, as desired. For example, video 3 is always mis-classified: 11 times out of 17 as video 2, and the remaining 6 times as video 6. Note that the mis-classification, in general, is not symmetric; that is, if video j is classified as video i most of times (with $i \neq j$), it does not mean that the same happens for video i. This characteristic is typical of WF mimicry defences, such as W-T, where each web page is strictly associated to a fake one. In Divergent, instead, it is a matter of clustering the resources inside the web server: we talk about this in the next chapter.

[2] A matrix is diagonal if and only if it is both upper- and lower-triangular, i.e., the entries outside the main diagonal are all zero

	Confidence VideoX									
VideoX	Video1	Video2	Video3	Video4	Video5	Video6	Video7	Video9	Video10	Video11
Video3	0.239	0.064	**0.356**	0.011	0.12	0.019	0.017	0.045	0.064	0.065
Video4	0.002	0	0	**0.473**	0.215	0.005	0.23	0	0.054	0.02
Video7	0	0	0	0.443	0.053	0.004	**0.498**	0	0.01	0
Video5	0.108	0.109	0.103	0.083	**0.126**	0.06	0.092	0.099	0.118	0.102
Video2	0.132	**0.157**	0.152	0.05	0.082	0.037	0.06	0.135	0.118	0.076

Table 6.1: Output probabilities of the adversarial CNN model against five different videos with Divergent

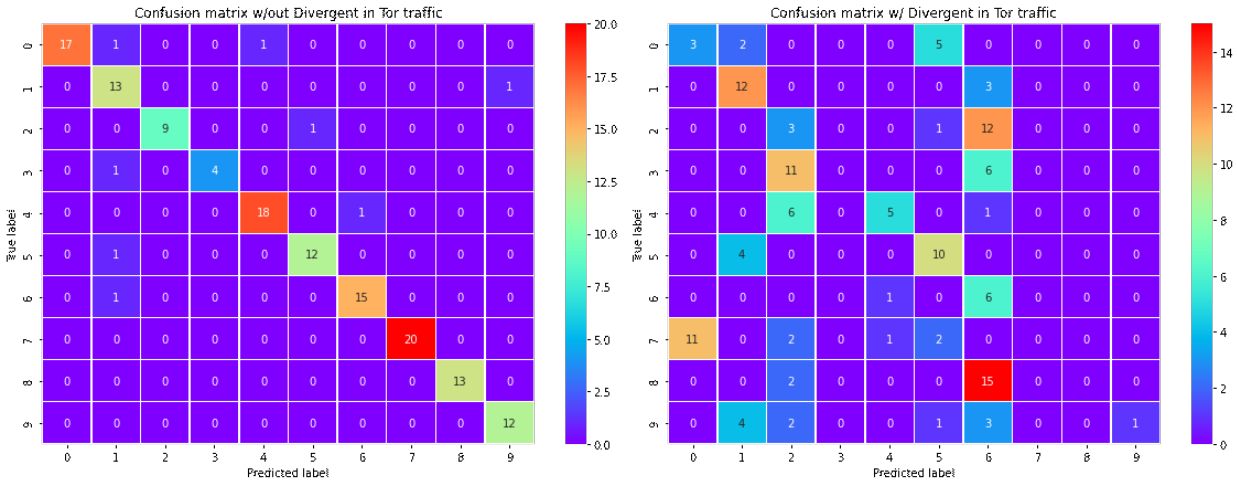


Figure 6.1: Comparison confusion matrices without Divergent (left) and with defence (right)

7

Discussion & future work

The results that we obtained show that deep learning approach is a powerful tool for WF attacks. However, defended traffic proves that Divergent provides a significant circumvention against the described offensives, saving bandwidth, and time, over other proposed defences. These results also bring to light a potential weakness in current defence methodologies. As regards Walkie-Talkie, the main menace is the inference threat, due to the fact that only one decoy page is associated to a real web resource, besides the considerable issue that the client must know a priori information about the resources to be loaded. In Divergent, instead, we tried to leverage the traffic shape, so that more and more resources appear to have the same network fingerprinting. To this aim, some possible improvements can be applied to the defence.

- *Clustering.* A plausible problem we can face is the following one: suppose that a client requests a very small resource-video (e.g., $10kB$ in size), and inside the web-server there exist videos which are way bigger (e.g., $1GB$). Of course, by simply introducing some 'noise' inside the transmission (that is, burst of dummy packets), we cannot decrease the performances of the adversarial model, unless we introduce a lot of overhead. A possible approach to bypass such hindrance is that, generally speaking, resources in the web server are clustered with other ones, so that elements in the same cluster will have a *similar* traffic flow, once they are requested. We thought on two different ways to cluster. The first one, the more explicit, is by exploiting real machine learning algorithms, such as k-means++, that aims to partition n observations into k clusters, in which each observation belongs to the cluster with the nearest mean (also called *centroid*), serving as

a prototype of the cluster. In this case, the 'observations' can be the actual resources or the traffic flow tight to the resource. Another possible mean, is by exploiting the state-of-the-art adversarial model, and its softmax activation function. In this way we can obtain the confidence of the output model (like we did and represented in table 6.1); so, videos with very similar traffic flow, will receive a similar confidence score as result of the softmax function, and will be placed in the same cluster. After clustering, similar resources will, probabilistically speaking, be assigned to similar traffic flows, e.g., by employing similar parameters in the binomial distributions (see Algorithm 5.1), when they must be sent to the client. In this way, the attacker, even if he could be able to grasp the right cluster, he will have less confidence in order to recognize the specific resource requested by the client.
For example, as we can see in the confusion matrix on the right of the plot 6.1, video 1 (the smallest one) and video 4 (the biggest one), have few mis-classifications with respect to all the others, due to what we just explained. In particular, in order to not introduce too much overhead, we assigned to video 1 a distribution that does not insert too many dummy packets. At the same time, we paired video 4 with video 2 (the second biggest resource), so that the model will have uncertainties on which video the victim is receiving (note again the non-symmetric property that we mentioned in section 6.3). All the other videos have been clustered by size-similarity. Therefore, we proved that by analyzing a good strategy for clustering the resources, we may further improve our defence.
This idea of clustering data is similar to a work made by Nithyanand R., Cai X. and Johnson R. [25], in which they propose *Glove*, an SSH-based defence that consists of two different phases. During the first phase, called *training* phase, the different network traces are collected, clustered according to their network-level features and finally a transformation matrix is generated. This matrix can convert one traffic distribution into another one, and it's exploited in the second phase, named *defending* phase.

- *Optimization.* As we discussed in chapter 5.2, the values assigned to x and y are fundamental for the overhead introduced in the communication. Moreover, they depend on the binomial PMF, which in turn relies on the size of the resource that the server is transmitting. The values that we assigned to the parameters of the binomial distribution are empirical, based on a specific analysis of the 10 videos we had as a sample. We tried to select the videos such that covered corner-stone scenarios, too, such as a very big and a very small resource as compared to all the other videos, and a heterogeneous setting.

  A possible way to improve Divergent, and make it more general, would be an extension of the previous improvement (clustering): based on the clustering and on the silhouette score, compute x and y that reduce at most the overhead. The *silhouette score* is a metric used to compute how good the resources have been clustered by a certain algorithm. Its value belongs to the interval $[-1, 1]$, and the bigger it is, the better the clustering has been performed. With this metric, we can understand how much the data inside the clusters are (dis-)similar and in this way opt to use small or significant values of the parameter n

of the binomial distribution.
In addition, we discuss about the possibility of computing the different values of y's during a pre-phase, instead of at runtime, in order to reduce the obstruction to the Quality of Experience (QoE), given by cost operation of sampling.

- *Open-world scenario.* In our work we supposed that the victim is only able to request specific resources and the attacker knows all of them. This type of setting is named as *close-world* scenario. A more realistic background instead supposes that the victim can ask for whichever data and the attacker, due to the huge amount that composes the web, monitors only specific web resources and all the others are labeled as *unmonitored*. In order to comprehend if the defence could be deployed also in a real-world scenario, an analysis in the context should be carried out. At the same time, we argue that the closed-world scenario is more convenient for an attacker, so we are confident that Divergent can work in this setting, too.

8 Conclusion

In this work we propose an innovative possible way to implement and apply a defence against traffic analysis, specifically the website fingerprinting ones. Although a significant amount of previous proposals has investigated the topic of preventing website identification attacks, some enhancements are still needed in terms, above all, of overhead and QoE of the user. Some approaches, even if are very efficient against the attacks, introduce too much useless data in the communication, others instead assume difficult-to-realize suppositions. The approach that we take into consideration, instead, ensures that a good defence efficiency can be achieved, while still keeping low the overhead. We accomplish this through the use of randomness, which allows to continuously change the traffic fingerprinting associated to each particular data[1], distinct probability mass functions per resource and, possibly, clustering. Moreover, our method does not need to introduce decoy pages in the server, increasing the wastefulness of memory server-side, nor that the client knows in advance the characteristics of the future loading web pages.

[1] That is the reason why we decided to name our defence *Divergent*, as different transmissions of the same resource *diverge* from each other

References

[1] (2022) Tls 1.2 vs 1.3 - the improvements you can expect. [Online]. Available: https://www.f5.com/c/landing/encrypted-threats/article/tls-1-3-are-you-ready-for-the-update

[2] B. Lee. (2022) Kerberos important changes. [Online]. Available: https://www.starwindsoftware.com/blog/active-directory-kerberos-changes-and-new-azure-active-directory-kerberos-feature

[3] Microsoft. (2021) Update adds aes encryption protections to the ms-samr protocol. [Online]. Available: https://support.microsoft.com/en-us/topic/kb5004605-update-adds-aes-encryption-protections-to-the-ms-samr-protocol-for-cve-2021-33757-e4da

[4] S. Pournaghi, B. Zahednejad *et al.*, "Necppa: A novel and efficient conditional privacy-preserving authentication scheme for vanet," 2018.

[5] A. Compagno, M. Conti *et al.*, "Don't skype & type! acoustic eavesdropping in voice-over-ip," 2017.

[6] R. Schuster *et al.*, "Beauty and the burst: Remote identification of encrypted video streams," 2017.

[7] A. Panchenko *et al.*, "Website fingerprinting at internet scale," *In Proceedings of the 23rd Network and Distributed System Security Symposium*, 2016.

[8] T. Wang *et al.*, "Effective attacks and provable defenses for website fingerprinting," *In Proceedings of the* 23rd *USENIX Security Symposium*, 2014.

[9] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," 2016.

[10] P. Sirinam *et al.*, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," 2018.

[11] T. Wang *et al.*, "Walkie-talkie: An efficient defense against passive website fingerprinting attacks," 2017.

[12] D. Hasselquist *et al.*, "Twitch chat fingerprinting," 2022.

[13] A. Panchenko *et al.*, "Website fingerprinting in onion routing based anonymization networks," *In Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, 2011.

[14] C. Wright *et al.*, "Traffic morphing: An efficient defense against statistical traffic analysis," 2009.

[15] K. P. Dyer *et al.*, "Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail," 2012.

[16] G. Jiaxi *et al.*, "Walls have ears: Traffic-based side-channel attack in video streaming," 2018.

[17] (2022) How many people use tor. [Online]. Available: https://truelist.co/blog/tor-stats/

[18] M. Perry. (2011) Experimental defense for website traffic fingerprinting. [Online]. Available: https://blog.torproject.org/experimental-defense-website-traffic-fingerprinting/

[19] Y. LeCun *et al.*, *Deep Learning*. Nature, 2015. [Online]. Available: https://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf

[20] ——, "Gradient-based learning applied to document recognition," 1998.

[21] G. Ian, B. Yoshua, and C. Aaron, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[22] X. Luo, *et al.*, "Httpos: Sealing information leaks with browser-side obfuscation of encrypted flows," 2011.

[23] W. De la Cadena, A. Mitseva, J. Hiller, J. Pennekamp, S. Reuter, A. Panchenko *et al.*, "Zero-delay lightweight defenses against website fingerprinting," *In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.

[24] J. Gong and T. Wang, "Zero-delay lightweight defenses against website fingerprinting," *In Proceedings of the 29th USENIX Conference on Security SymposiumAugust 2020*, 2020.

[25] R. Nithyanad, X. Cai, and R. Johnson, "Glove: A bespoke website fingerprinting defense," *In Proceedings of the* 13*th Workshop on Privacy in the Electronic Society*, 2014.

[26] M. Juarez *et al.*, "Wtf-pad: Toward an efficient website fingerprinting defense for tor," 2016.

[27] A. Mehta *et al.*, "Pacer: Comprehensive network side-channel mitigation in the cloud," 2022.

[28] (2015) Privileges tcpdump. [Online]. Available: https://askubuntu.com/questions/530920/tcpdump-permissions-problem

[29] Wireshark website. [Online]. Available: https://www.wireshark.org/

[30] S. J. Murdoch *et al.*, "Low-cost traffic analysis of tor," 2005.

[31] (2022) How many people use vpn. [Online]. Available: https://www.security.org/vpn/statistics/

[32] (2022) How many people use internet. [Online]. Available: https://earthweb.com/how-many-people-use-the-internet-daily/

[33] Z. Gutterman *et al.*, "Analysis of the linux random number generator," 2006.

[34] Y. Theodore. (2020) [git pull] random: changes for 5.6. [Online]. Available: https://lore.kernel.org/lkml/20200131204924.GA455123@mit.edu/

[35] M. Larabel. (2020) /dev/random is more like /dev/urandom with linux 5.6. [Online]. Available: https://www.phoronix.com/news/Linux-5.6-Random-Rework

[36] Google. Homepage tensorflow. [Online]. Available: https://www.tensorflow.org/

[37] ——. Homepage keras. [Online]. Available: https://www.keras.io/

[38] ——. Homepage colab. [Online]. Available: https://colab.research.google.com/

[39] J. Lokoc, B. Münzer, K. Schoeffmann, M. Del Fabro, M. J. Primus, T. Skopal, and J. Lánský, "What are the salient keyframes in short casual videos? an extensive user study using a new video dataset," *Proceedings of IEEE International Conference on Multimedia Expo Workshops (ICMEW 2015)*, 2015.

Acknowledgments

These five years have been a long, great journey, during which I met a lot of people, I had the possibility to grow up as a person both behaviourally and intellectually.
First of all, I would like to thank my family: my mum, Barbara, my dad, Federico, my little brother, Leonardo (also called LeoBeo), who always believed in me and my beloved girlfriend, Silvia, who, thanks to her strength, endless joy and smile always helped me, above all during the hardest moments.
I want also to acknowledge my friends, because even if we had not the possibility to meet each other a lot during this period, always remained and persisted in keeping in touch and stay by my side.
Last but not least, I want to express my huge gratitude to my supervisor, Prof. Mauro Conti, and co-supervisor, Dr. Chhagan Lal, who helped me to make possible this work, despite all the difficulties and the time spent in finding the right topic.

To all of you, thank you.
Estote parati.