TÉCNICO LISBOA

Deanonymization of Tor Onion Services with Acceleration-Based Watermarking

Afonso Manuel Vieira Machado Barros de Carvalho

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor(s): Prof. Nuno Miguel Carvalho dos Santos
Prof. Diogo Miguel Barrinha Barradas

Examination Committee

Chairperson: Prof. Luís Manuel Antunes Veiga
Supervisor: Prof. Diogo Miguel Barrinha Barradas
Member of the Committee: Prof. Henrique João Lopes Domingos

November 2022

Acknowledgments

I would like to begin by thanking my advisors, professors Nuno Santos and Diogo Barradas, for the continued and relentless support and motivation given to me during this process. I would like to thank my family for all their love and affection; for always giving me everything I needed to succeed and for always having my back no matter the path I choose to follow. I would like to thank my friends, and specially my girlfriend, for keeping me sane and making my life more colorful. I would like to thank all the teachers I ever had as I would not be the person I am today without them. Finally, I would like to dedicate this to my late uncle, who I love and miss dearly.

Resumo

A rede Tor e a sua infraestrutura de Serviços Onion (SOs) permitem que utilizadores usem e prestem serviços na Internet mantendo o anonimato, quer do lado do emissor, quer do recetor. Isto permite-lhes contornar filtros de censura e manter-se a salvo de repercussões, mas também faculta uma plataforma para a prática de atividades ilícitas. Isto fez com que o Tor se tornasse um alvo atraente para atacantes, incluindo autoridades policiais (APs) que procuram identificar SOs criminosos. Estas APs podem colaborar entre si, ganhando amplo acesso a informação relativa a tráfego de rede recolhida ao nível de Sistemas Autónomos ou de Pontos de Presença dos operadores de telecomunicações. É sabido que esta informação permite que sejam lançados ataques de deanonimização à infraestrutura do Tor.

Ataques recentes mostram que é possível a um atacante global passivo quebrar por completo o anonimato de sessões de SOs do Tor com alta exatidão. No entanto, as técnicas propostas implicam a monitorização de um grande número de fluxos Tor, o que introduz constrangimentos não só relativos à escalabilidade mas também ao sucesso destes ataques. Neste projeto, propomos um ataque alternativo, o DissecTor, que faz uso não só de uma nova técnica de marca de água baseada em aceleração mas também de aprendizagem automática para deanonimizar um SO específico de forma eficaz e disfarçada. Identificando o SO-alvo, consegue reduzir-se significativamente a complexidade e os recursos necessários para deanonimizar a sessão completa. Realizamos uma avaliação profunda do sistema DissecTor, exploramos diversas variantes e refletimos acerca da sua utilização.

Palavras-chave: Tor, correlação de tráfego, serviços onion, marca de água, aprendizagem automática

Abstract

The Tor anonymity network and its Onion Service (OS) infrastructure allow users to browse and provide services on the Internet while benefiting from sender and receiver anonymity. This enables them to circumvent censorship filters and remain safe from prosecution but also provides a means to conduct illegal activities. This has made Tor an enticing target for attackers, including Law Enforcement Agencies (LEAs) seeking to identify unlawful OSes. These LEAs may form coalitions, gaining broad access to network traffic information collected at the level of Autonomous Systems or Internet Exchange Points. This information is known to allow deanonymization attacks on Tor's infrastructure to take place.

In fact, recent attacks have shown that it is possible for a global passive adversary to fully deanonymize Tor Onion Sessions with high accuracy. The proposed techniques, however, depend on the monitoring of a large amount of Tor flow samples introducing serious bottlenecks to the scalability of these attacks and to the maximum achievable recall (i.e., the number of sessions that can effectively be screened by the adversary). In this work, we propose an alternative attack, DissecTor, that utilizes a new acceleration-based watermarking scheme and machine learning techniques to deanonymize a targeted OS both effective and covertly. By identifying the target OS one significantly reduces the complexity and resources needed to fully deanonymize the whole browsing session. We performed an in depth evaluation of the DissecTor system, exploring several variants and reflecting on the best use cases for each.

Keywords: Tor, traffic correlation, hidden services, watermarking, machine learning

Contents

Acknowledgments . . . iii
Resumo . . . v
Abstract . . . vii
List of Tables . . . xi
List of Figures . . . xiii

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Goals . . . 2
1.3 Contributions . . . 3
1.4 Thesis Outline . . . 3

2 Background and Related Work **5**
2.1 Background on Tor . . . 5
2.2 Traffic Correlation Attacks . . . 9
2.3 Watermarking Techniques . . . 15
2.4 Summary . . . 21

3 DissecTor **23**
3.1 System Overview . . . 23
3.2 Acceleration-Based Watermark . . . 26
3.3 Watermark Amplification . . . 29
3.4 Watermark Classifiers – Training and Detection . . . 30
3.5 Anomaly Detection . . . 31
3.6 Implementation . . . 32

4 Evaluation **35**
4.1 Methodology . . . 35
4.2 Watermark Detection . . . 36
4.3 Variants . . . 40

5 Conclusions **51**

5.1 Achievements 51

5.2 Future Work 52

Bibliography **53**

List of Tables

2.1 How DissecTor compares to the studied watermarking systems. 21

4.1 Detection metrics when performing watermark detection with the C4.5 classifier. 38
4.2 Detection metrics for watermark detection with OCSVM using the default parameters. . . 40
4.3 Detection metrics when performing watermark detection with the Naive Bayes classifier. . 41
4.4 Detection metrics when performing watermark detection with the Random Forest classifier. 41
4.5 True and False positive rates when performing watermark detection with OCSVM for all default parameters after pre-processing the datasets with standard scaling. 44
4.6 True and False positive rates when performing watermark detection with OCSVM using all default parameters except *Gamma* which was set to “auto” instead of “scale”. 44
4.7 Best parameter setup for each PCR using the Reply stream according to the F1-score metric. 47
4.8 Best parameter setup for each PCR using the Fetch stream according to the F1-score metric. 47
4.9 Best parameter setup for each PCR using the Joint stream according to the F1-score metric. 47
4.10 Best parameter setup for each PCR using the Reply stream according to the P4 metric. . 48
4.11 Best parameter setup for each PCR using the Fetch stream according to the P4 metric. . 48
4.12 Best parameter setup for each PCR using the Joint stream according to the P4 metric. . . 48

List of Figures

2.1 Three relay Tor circuit establishing connection between Alice and Bob. 6
2.2 Onion Service session between a client, Alice, and a hidden service, HS. 8
2.3 A Tor circuit between Alice and Bob spanning several ASes. 9

3.1 DissecTor's architecture and main component interactions 24
3.2 DissecTor's pipeline 25
3.3 Top: Size of network packets recorded from the reply stream of a target OS. Bottom Left: Grouping the recorded packet sizes with G=1s. Bottom Right: Grouping them with G=3s. 27
3.4 Stream throughputs derived from the packet size data from Figure 3.3. On the left G=1s and on the right G=3s. A cleaner signal can be observed with a coarser time interval (G=3s). 28
3.5 Stream accelerations derived from the throughputs on Figure 3.4. Once again, on the left G=1s and on the right G=3s 29

4.1 True and False positive rates for all three classifiers: C4.5, Naive Bayes and Random Forest. 42
4.2 Best F1-score and P4 values of all three classifiers per PCR. 42
4.3 Impact of the "Nu" parameter on the true and false positive rates of DissecTor running with OCSVM in the otherwise benchmark configuration. 45
4.4 Impact of the "Tolerance" parameter on the true and false positive rates of DissecTor running with OCSVM in the otherwise benchmark configuration. 46
4.5 Best F1-score and P4 values of all three streams per PCR when using OCSVM. 49
4.6 Comparison between the best F1-score and P4 values of both versions of DissecTor. . . 49

Chapter 1

Introduction

This thesis presents a new attacking technique against the Tor anonymity network. In our work, the general approach that we employ focuses on the usage of traffic correlation attacks to break the anonymity guarantees of Tor, specially its Onion Service (OS) infrastructure. We propose a new technique based on active traffic analysis that relies on flow watermarking and machine learning to identify OSes by their IP addresses. We will not only present a detailed description of the system carrying out said attacks but also describe the results of a thorough evaluation of the system's performance and effectiveness.

1.1 Motivation

Tor is a popular anonymity network that stemmed from the belief that "Internet users should have private access to an uncensored web" [1]. Tor's anonymity guarantees have made it a fundamental tool for promoting free speech and protecting the privacy of Internet users all over the world. In fact, Tor has helped a vast range of users – ranging from journalists and political activists to whistleblowers and citizens living in repressive regimes – to browse the Internet privately and enabling such users to circumvent censorship filters and to remain safe from prosecution [2–4].

Tor is built on top of the onion routing protocol [5, 6], whereby a message is encapsulated in successive layers of encryption and transmitted along several relay nodes before arriving at its final destination. Each node that receives the message is able to decipher the top-most layer of encryption in order to uncover the data's next destination. Only the last node can decipher the message's contents. This means that each intermediate node in the chain, called a circuit, knows only the identity of its predecessor and successor, keeping the sender's anonymity intact.

In Tor, this principle is not limited to clients. Tor's Onion Services (OSes), previously called Hidden Services (HSes), are websites or services that use the Tor technology to keep their identities (i.e., IP addresses) secret. Unlike "normal" websites, where the client knows the service's IP and the message is directly sent to it, an OS's address is kept hidden from the client. A rendezvous node is used as intermediary for the message exchange and both the client and OS communicate with it using onion routing in the form of two separate Tor circuits.

Due to its strong anonymity properties, Tor has since become an interesting target for state-level actors. For instance, repressive governments are interested in breaking Tor to prevent abuse and political dissidence [7], while law enforcement wish to fight organized online crime [8–12]. Many attacks have been developed in the twenty years elapsed since the Tor network's initial launch, in October 2002. According to the taxonomy proposed by Evers et al. [13] attacks on the system can be grouped in seven categories: *correlation*, *congestion*, *timing*, *fingerprinting*, *denial of service*, *supportive*, and *revealing hidden services*. They can also be classified as *passive* or *active*, if the attacker only observes traffic or has the ability to manipulate it, respectively, and as *single-end* or *end-to-end*, if the attacker monitors/controls only one of the edge relays of a Tor circuit or both simultaneously.

One of the most significant threats to Tor is that of correlation attacks [14], end-to-end attacks where an adversary monitors both edges of a Tor circuit and seeks to establish a relation between flows observed at the entry and exit points. Since the entry relay knows the sender and the exit relay knows the receiver, sound correlations can effectively deanonymize the communications pertaining to that specific Tor circuit and enable the monitoring of a user's activity. To make matters worse, the success of passive correlation attacks can be significantly improved through the usage of active watermarking techniques [15]. Put briefly, an attacker can carefully manipulate the traffic patterns of specific Tor circuits (e.g., introduce predictable delay on packets) to ease correlation efforts.

A recently proposed correlation attack on Tor leverages a system called Torpedo [16]. Torpedo seeks to enable a global passive adversary to deanonymize Tor onion service sessions, i.e. deanonymize OSes and their users and establish correlation of the latter's browsing sessions. Torpedo collects traffic data from Autonomous Systems (ASes) and uses it to establish correlation of past events. The collected data pertains to Tor flows originated at or destined to Tor relay nodes, of which a public list is available. To increase the chance of correlation, Torpedo includes a network traffic flow filtering pipeline and a set of heuristics that trim down the possibilities of successfully correlated flows. This process enables Torpedo to correlate Tor flows with very high precision. In spite of its precision, Torpedo is seldomly successful and achieves low coverage, missing the majority of correlated flows. This is a product of Torpedo's filtering and correlation steps being both very expensive from a computational point of view and poorly scalable, specially when considering the potential $N \times M$ relation of N users and M OSes.

1.2 Goals

Following up on the work that resulted in the development of Torpedo, the goal of this thesis is to investigate an alternative attack that aims specifically to deanonymize a targeted OS (as opposed to deanonymizing both the client- and the server-side of OS sessions). Assuming that the attacker can control the client-side and generate arbitrary traffic directed towards a predefined OS, we speculate that it is possible to launch correlation attacks to deanonymize Tor Onion Services with higher coverage and scalability than Torpedo's. By successfully identifying the target Onion Service's address one significantly reduces the complexity and resources needed to fully deanonymize the whole browsing session using a solution like Torpedo's. This stems from the reduction of the problem's search space, previously

$N \times M$, to N where N and M are the number of users and the number of OSes respectively.

To achieve this goal, the proposed solution will leverage active attacks based on watermarking combined with state-of-the-art traffic correlation techniques, such as recent attacks based on deep neural networks [14, 15]. The central technical challenge lies in finding a watermarking technique that can simultaneously generate a watermark that (a) can be accurately detected at the ingress link of the targeted OS, and (b) is stealthy enough to prevent being detected by the OS provider. A thorough study of the state-of-the-art in both watermarking and traffic correlation must be pursued, followed by a careful adaptation and testing phase of the most promising techniques at our disposal. The resulting specification must then be evaluated through extensive experiments where both accuracy and stealthiness will have high priority. We will assess the true positive and false positive rates of the solution for different configurations of the system's pipelines and varying user and OS number, activity and location.

1.3 Contributions

In this work we present a new system called DissecTor able to deanonymize specific Tor Onion Services. DissecTor leverages an active approach, relying on flow watermarking and machine learning, in order to reveal the target's identity in the form of an IP address. Specifically, the technical contributions of this thesis can be summarized as follows:

- The design and implementation of DissecTor, which can launch active attacks to targeted OSes in two different stages: (1) send watermarked traffic to the target of which an onion address is known, and (2) identify the watermarked connection between the OS and its guard node from a pool of candidates using machine learning techniques.
- The development of a novel watermarking technique named *acceleration-based watermarking* which allows for deanonymizing OSes by identifying predefined variations in the receiver's traffic based on perturbations introduced by probes sent by the attacker.
- An extensive evaluation of our system which covers two different watermark detection techniques and is mainly based on finding accuracy for a vast set of parameter variations. Besides comparing both techniques we discuss how varying certain parameters might compromise the stealthiness of the attack and reflect on this trade-off.

1.4 Thesis Outline

The remainder of this thesis is structured as follows: Chapter 2 presents all the background and related work. Chapter 3 describes the developed solution's design and implementation as well as the watermarking and machine learning techniques employed in it. In Chapter 4, we present the results of DissecTor's evaluation. This includes two different types of watermark detection methodologies, each with several variants, and an analysis of the impact of the watermark signature on its detection and applicability. Finally, Chapter 5 concludes the thesis and presents possible directions for future work.

Chapter 2

Background and Related Work

In this chapter we present the relevant background that served as the basis for the work carried out on DissecTor. Since the objective of our work consists in the development of a system capable of conducting deanonymization attacks on the Tor Onion Service infrastructure, it is essential to cover how Tor works, how the system was extended with the introduction of Onion Services and how it has been, and still is, the target of a multitude of attacks (Section2.1). In this regard, and since the attack we developed is heavily influenced by both flow watermarking techniques and traffic correlation attacks in general, we will have sections dedicated to both topics in this chapter (Sections 2.2 and 2.3).

2.1 Background on Tor

In this section, we provide a brief introduction to the Tor anonymity network. We begin by explaining how sender anonymity is preserved through the usage of onion routing and present the system components that make it possible. We then explain how the concept of sender anonymity was expanded to include the receiver and introduce Onion Services as well as the rest of the infrastructure that makes it feasible.

2.1.1 Sender Anonymity

Tor [17] is a distributed overlay network designed to anonymize low-latency TCP-based applications. To do so, Tor implements its own version of the *onion routing protocol* [5, 6]. In onion routing, messages reach their destination through a series of intermediaries called *relays* or *routers*. Before being sent, a message is wrapped in multiple layers of encryption to protect both its contents and the route it will take. When reaching the next relay of the route, the top-most layer of encryption is peeled, giving the router access to the information it needs to properly forward the message. Although essential for this system to work, relays are not completely trusted, and are only given access to the previous and next hops along the route. This ensures the system remains secure in case any particular router is compromised. In the case of Tor, the connection established between sender and receiver is called a *circuit*. Typically a Tor circuit is three relays long. These are called the *entry*, the *middle* and the *exit*, respectively in the direction from sender to receiver. All onion routers communicate with each other and with clients

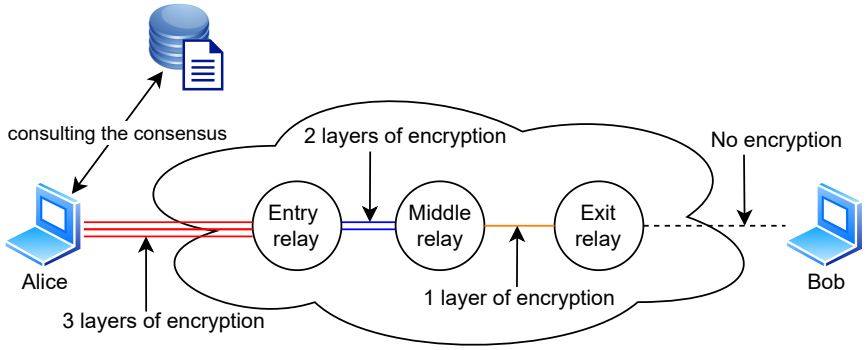

Figure 2.1: Three relay Tor circuit establishing connection between Alice and Bob.

through TLS connections and all traffic passes along said connections in fixed-size cells of 512 bytes. This keeps individual packet sizes secret and further adds to the system's privacy guarantees.

When creating a new circuit, a user first obtains a list of the currently-running relays in the Tor network. This document is called a *consensus* and is compiled and voted on by the *directory authorities*, special-purpose and trusted relays whose job is to keep track of all available relays, once per hour. When a set of relays is chosen out of the consensus, the user negotiates symmetric keys with each one, building the circuit incrementally, one hop at a time. In other words, negotiation of the first key takes place with the first relay and, if successful, a new one-hop circuit is established. After this, further negotiations are carried out through this circuit using onion routing. Each time a negotiation is successful a new relay is added at the end of the circuit as it grows to reach its final length.

As mentioned above, all messages sent through the circuit are successively encrypted according to the number of relays in the circuit. For a three relay circuit, the client begins by encrypting the message with the exit relay's key, followed by the middle's key and finally the entry's, as the last layers, the most superficial, will be the first to be removed. When the message arrives at the exit relay it peels the last layer of encryption, establishes a TCP/IP connection to the destination and forwards the original packets from the sender, serving as its proxy. Only the entry relay knows the IP address of the sender. All others, including the receiver, remain oblivious to it. Figure 2.1 presents an example of a three relay Tor circuit allowing Alice, the sender, to communicate with Bob, the receiver. It also shows how Alice had to consult the consensus before building the circuit, in order to know which relays were available, and the amount of encryption layers around the message at each of the hops.

2.1.2 Receiver Anonymity

The aforementioned protocol allows for sender anonymity. It lacks, however, the ability to guarantee the same for the receiver, as its IP address must be known by both the sender and the exit relay for the communications to take place. A natural extension to the protocol was put in place by the Tor developers precisely to tackle this issue. The devised solution is called *location-hidden services*, most commonly known simply by *hidden services* (HSes) or *onion services* (OSes) [17]. This mechanism aims at allowing the operation of TCP services, such as HTTP, FTP, IMAP etc., without them having to publicise their IP addresses. Next, we describe the steps of this extended Tor protocol to establish

mutually anonymous sessions between a client and a hidden service.

1. Firstly, a hidden service (HS) generates a long-term public key pair to identify itself. The public key is used to generate the service's onion address, which a user may come across and use when trying to reach out. It then selects a set of relays to use as introduction points. To this effect it begins by connecting to them using two-relay Tor circuits and asks them to act as such. If the answer is affirmative a long-term Tor circuit is established to the introduction points. Note that they do not know the HS's address.

2. The next step is to publicly advertise the introduction points so users can contact them. This is done by compiling a list of these relays, signing the list with the private key of the pair mentioned above and uploading it to a distributed hash table that acts as a lookup service.

3. When a client wishes to access the HS it needs to first retrieve its details from the lookup service using Tor (in order to maintain anonymity). Once the introduction point list is retrieved, its signature is validated using the service's onion address which, as said before, contains the public key of the service. Once the client establishes that the data retrieved from the lookup is correct it chooses a relay to serve as *rendezvous point* (RP). To this end, a two-relay Tor circuit is established to the RP and a one time secret, also referred to as *rendezvous cookie*, is sent to the RP for future use.

4. The client then proceeds to open an anonymous stream to one of the HS's introduction points and shares information regarding itself, the RP, the rendezvous cookie and the start of a Diffie-Hellman handshake (all messages are encrypted using the HS's public key). This information reaches the HS that then decides on whether to connect to the client or not.

5. The final step is for the HS to connect itself to the RP. It does so using a three-relay Tor circuit. When a connection is finalized the HS sends the rendezvous cookie and the second part of the Diffie-Hellman handshake. If the RP determines that the rendezvous cookie is valid the process is concluded. From that point on all messages from both the sender and the receiver are sent to the RP that forwards them accordingly. It is important to note that the RP never knows the sender IP, the receiver IP or the content of the exchanged messages, assuring the anonymity of both parties.

Figure 2.2 depicts the entirety of the process we just described with a few simplifications. In the figure we can see HS, a hidden service, maintaining a permanent two-relay Tor circuit, in blue, with its sole introduction point (IP). We can also see its connection to the Lookup Service, which is used to publish the signed list of introduction points. On the other hand we see Alice, the client, requesting said list from the Lookup Service and connecting to the rendezvous point, RP, through a two-relay circuit in red. After connecting to RP, Alice proceeds to "introduce" herself to HS via IP. After IP sends Alice's request to HS and the former agrees to a connection, the green circuit is created, linking HS to RP and closing the final six-relay Tor circuit between Alice and HS. The two parties can now exchange messages normally with the assurance that their identities are kept secret from one another and from anyone besides relay R9, that knows Alice's address, and relay R14, that knows HS's.

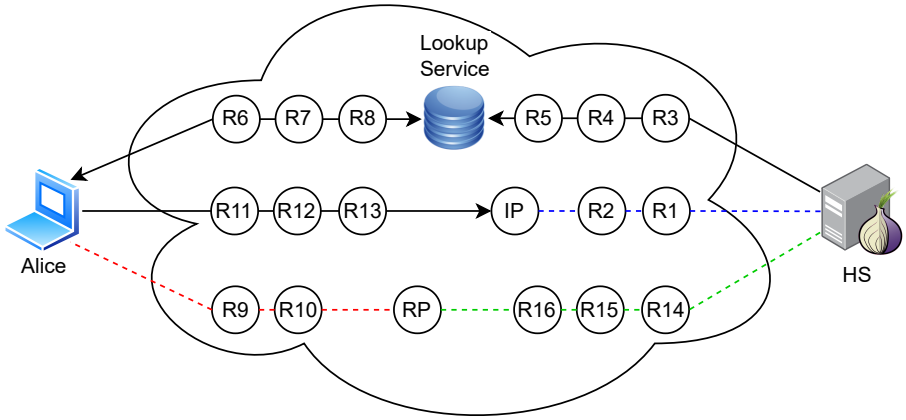

Figure 2.2: Onion Service session between a client, Alice, and a hidden service, HS.

2.1.3 The Security of Tor

Tor's anonymity guarantees stem from the fact that it is extremely hard for a single entity to control both the entry and exit relays of a given circuit. If that were the case, an attacker would have access to both sender and receiver addresses and, although no messages could be observed being sent directly from one to the other, traffic correlation techniques could be used to deduce their interaction with high probability, using inter-packet arrival times and volume.

In fact, it is not even necessary for an entity to control a relay, but only to be able to observe its traffic. In that sense, Internet Service Providers (ISPs) and, more broadly, Autonomous Systems (ASes) inherently find themselves in a privileged position to launch such deanonymization attacks on the Tor network. An AS is a large network, or network of networks that share the same routing policy, and are managed and maintained by the same entity of set of entities. These can be ISPs, other technology companies, governments, universities, amongst others. The Internet as we know it is nothing but a network of these large networks called ASes.

For an ISP or AS to deanonymize a Tor circuit it is still necessary for them to access traffic information of both the its entry and exit relays, which in practice is difficult due to Tor's tendency to select relays connected to different such entities. In practice, assuming both ends of the circuit are connected to different ASes altogether, a correlation attack would only be feasible if (a) an intermediary AS was responsible for forwarding the traffic of both these relays to their destinations or (b) an agreement was struck between two ASes that forward traffic from each of the relays to share information. Scenario (a) can prove technically and financially challenging due to the sheer amount of data flowing through an AS at any given time, which would require a considerable and expensive amount of processing power to analyse. Scenario (b), although seemingly more practical, would also prove difficult given inter-AS competitiveness, user privacy concerns, jurisdiction incompatibilities, foreign affairs, just to name a few.

Figure 2.3 depicts a realistic Tor circuit connecting Alice and Bob, spanning several ASes. This circuit has three relays, as is the norm for these types of circuits: R1, connected to AS5; R2, connected to AS9 and R3, connected to AS4. The image also shows several ASes that, although not directly

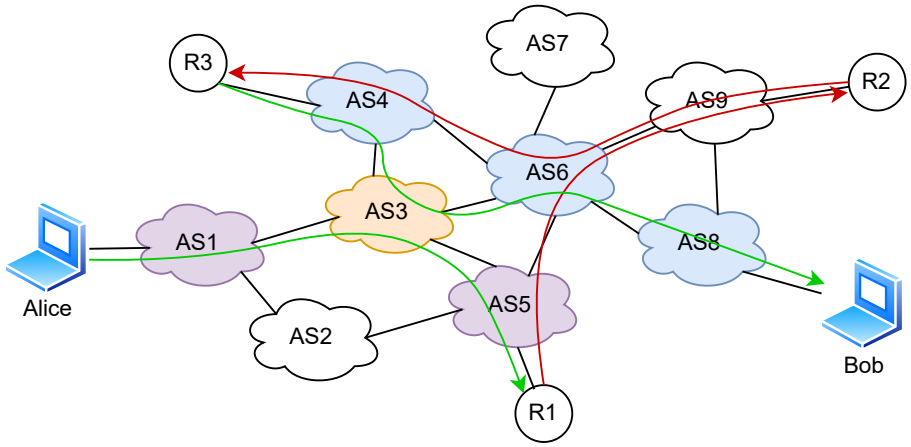

Figure 2.3: A Tor circuit between Alice and Bob spanning several ASes.

connected to any relays or clients, are necessary for the system to be viable, as they allow traffic to reach its destination when a direct path is otherwise nonexistent. Furthermore, we can observe a message being sent from Alice to Bob as well as the ASes it traverses along its route. Regarding the two scenarios mentioned above, we can see there is one AS capable of performing (a), AS3, as it forwards the message both from Alice to R1 and from R3 to Bob. Scenario (b) may occur if any one of the ASes that forward the message from Alice to R1 (AS1, AS3, AS5) collude with any one of the ASes that do so from R3 to Bob (AS4, AS3, AS6, AS8).

The following section seeks to introduce the relevant work present in the literature on which our attack will be based upon. We will mainly focus on correlation attacks directed at the Tor network as well as watermarking techniques, as these will be the base for our own system.

2.2 Traffic Correlation Attacks

In this section we present an overview of traffic correlation attacks. We begin by explaining some key concepts and then present several examples, going over each one briefly. These will be classified according to the used techniques and are introduced in approximate chronological order.

Traffic correlation attacks, also known as confirmation attacks, occur when an adversary attempts to link two network flows solely based on the observable features of the traffic and not its contents. This type of attacks is primarily used in settings where obfuscation techniques such as encryption are put in place. Correlation attacks may be passive or active, i.e., the adversary may simply observe the traffic or be able to manipulate it. Attacks also differ in respect to the means by which the adversary gains visibility into the network. This can be done by compromising or running Tor relays [18–20] or, regarding network-level threats like ASes and ISPs [21–23], by eavesdropping on the links they inherently have access to. The correlation itself may be done based on several different techniques such as pure statistical analysis, deep learning, traffic models, watermarking and others.

Given that Tor imposes a fixed cell size, this feature, which otherwise could be used to perform correlation, is not usable in this regard. Nonetheless, given that Tor seeks to maintain low latency, it

does not take significant measures to mask inter-packet timings or volume, and these features end up being used instead. Correlation attacks performed against Tor rely on flows captured at both ends of a circuit and do not need to know its full path. Since each edge relay knows the identity of one of the communicating parties, a successful correlation attack effectively results in the deanonymization of both the user's IP and the IP it is accessing.

Several such attacks have been proposed over the years, showcasing a number of different approaches, some of which having been mentioned before. In addition, a number of routing attacks have also been proposed [24, 25] and can be used to complement correlation by increasing an attacker's chances of intercepting the desired flows. We now present some of the correlation attacks proposed over time which made use of a vast array of techniques.

2.2.1 Timing-based

Some of the first traffic correlation attacks used purely statistical analysis of flow features to link ingress and egress portions of mix-network communication channels, such as Tor's circuits. These features include, but are not limited to, inter-packet arrival times, packet volume and bursting patterns. In 2004, Levine et al. [26] demonstrated, through the simulation of several mix-network systems, that timing attacks posed a significant challenge for those attempting to support low-latency applications over such systems. In particular, the paper focused on inter-packet timing information as a means to establish a connection between traffic observed at different relays of the system (like the entry and exit relays of a Tor circuit). The principle behind this idea is that if two relays are part of the same path then their inter-packet arrival times should be correlated in some way.

In 2005, Murdoch et al. [27] developed an attack directed at identifying the relays that make up a Tor circuit by selectively and successively congesting different routers while looking for increased transmission latency through circuits. The principle behind this active attack is based on the fact that a relay may be involved in the routing of several circuits simultaneously and that a highly congested relay will introduce noticeable latency on said circuits. If one can observe an increase in latency on a connection while congesting a particular relay, a conclusion may be drawn that said relay is part of the circuit's path. Although this attack can not reveal the participants' identities, it nonetheless compromises their anonymity. Murdoch et al.'s attack did not scale well, however, and became impractical as the Tor network kept growing. Work has been done on extending the attack in order for it to work on a larger scale. Evans et al.'s [18] 2009 proposal did exactly that, extending Murdoch et al.'s attack with a, at the time, novel bandwidth amplification attack. The underlying principle of correlation based on congestion and increased latency remained.

Around this time, Zhu et al. [28] also used correlation attacks based on traffic analysis to attack mix-based low-latency anonymous networks (as is the case of Tor), and found that all but a few proved vulnerable to their attacks. They considered seven different packet batching strategies and two implementation schemes and concluded that by using statistical analysis an adversary could accurately determine the output link of the traffic originated at a given input link when able to collect sufficient data.

2.2.2 Bandwidth-based

Until this point we mentioned attacks that based their correlation on timing features such as inter-packet arrival times. And although timing attacks had been proven successful, one requirement shared by many of them was either the involvement of a global adversary or the compromise of a vast number of relays. In 2010, Chakravarty et al. [19] introduced a new technique that would allow an adversary to identify the IPs of both Tor users and Tor Hidden Services. This technique was, unlike the ones mentioned above, based on available bandwidth estimation and not timing data and could be launched through a single colluding end-point. The attack relied on the ability of the colluding end-point to introduce traffic fluctuations towards the anonymous targets which would directly affect the available bandwidth on the circuit being used. These fluctuations would propagate through the network and the progress would be tracked using a bandwidth estimation tool called LinkWidth [29]. This attack was moderately successful, achieving true positive rates of around 49.5% in a controlled real-world setting when attempting to correlate Tor clients and Hidden Services. Furthermore, and although the authors concluded that the success of this attack was dependent on circuits' end-to-end throughput, the system proved these types of attacks could become a real threat to Tor in future.

2.2.3 Application-based

In 2011, Wang et al. [20] introduced a new, HTTP-based, application-layer attack. This was an active approach that required an adversary to be positioned at either end of a Tor circuit by operating malicious relays. Simply put, the exit relay would send a fake HTTP response towards the client with either a forged web page or a modified version of the original target web page which contained links corresponding to invisible objects. When the client received the malicious web page its browser would be tricked into initializing malicious connections in order to retrieve said objects and the traffic pattern that would ensue could be identified by the malicious entry relay by correlating it to an expected standard according to the malicious web page's characteristics.

2.2.4 Watermark-based

Some correlation attacks adopt a technique called *watermarking*, where an active adversary manipulates selected flows near their source so as to introduce identifiable signatures, know as watermarks, that can be easily identifiable by an observer. The watermarks may be carried by the manipulated flows in several different ways. There are methods for embedding watermarks that are content-based, time-cased, frequency-based and space-based. In the setting of anonymity networks, such as Tor, content-based techniques are not usable given the impossibility of accessing the flows' contents imposed by encryption schemes. Consequently, known watermarking attacks on such networks make use of the other types of carriers. Such is the case of Wang et al.'s [30], Houmansadr et al.'s [31] and Iacovazzi et al.'s [32] proposed systems, which we will explore in further detail in a later section along with others.

2.2.5 Prominent flow correlation attacks

Some correlation attacks merit a more in-depth explanation beyond their simple classification into the categories covered so far, whether they provided relevant and original insights or because they made use of state-of-the-art techniques.

Raptor

In 2015, Sun et al. [25] presented a set of attacks enabling AS-controlling adversaries to compromise user anonymity. The developed system, named Raptor, leveraged the dynamic aspects of BGP, the Border Gateway Protocol, and was divided into three individual attacks whose effects were compounded.

Asymmetric traffic analysis: The first attack introduced the notion of asymmetric traffic analysis. In other correlation attacks it was assumed that the attacker not only monitored the circuit's endpoints but also was able to observe traffic flowing in the same direction in each vantage point. In reality, given that Internet paths are often asymmetric, it might be the case that an AS-level adversary can see traffic flowing in different directions in both endpoints of the circuit. Raptor leveraged the fact that Tor and other anonymity systems use SSL/TLS to perform correlation using TCP sequence and acknowledgement numbers which are left unencrypted by the protocol. The authors showed this attack boasted a 95% accuracy when evaluated using the Spearman's rank correlation coefficient.

Take advantage of churn: A second insight conveyed in Sun et al.'s work consisted of exploiting natural churn to increase an AS-level adversary's surveillance capability over time. The authors concluded that the amount of Tor circuits a single AS performing asymmetric traffic analysis could compromise increased up to 50% over the period of one month. This increase was justified by churn and the network's consequent reconfiguration. Over time, the number of circuits whose endpoint connections crossed the attacker AS simultaneously in either direction increased by the previously mentioned amount.

BGP interception attacks: Lastly, the authors of Raptor demonstrated that Tor was vulnerable to BGP interception attacks by successfully performing one on the live network. In this attack, a malicious AS hijacked traffic otherwise directed at another AS by falsely advertising its ownership over a range of IP addresses owned by the legitimate one. By doing so, the adversary gained access to Tor traffic it could use to perform traffic correlation. Afterwards, in order to avoid interrupting the connection, the traffic was redirected back to its original path.

Raptor showed how the combination of several different techniques can increase an attack's effectiveness. The insights brought forth by the authors regarding asymmetric correlation and using churn to expand the surveillance capabilities of an attacker allowed for a system that boasted high accuracy even in otherwise less desirable conditions. Raptor's main drawback was the high observation volume needed to perform asymmetric correlation in such conditions.

DeepCorr

In 2018, Nasr et al. introduced DeepCorr [14], a flow correlation attack using deep learning. This attack was introduced as a response to the belief that most existing flow correlation attacks at the time were, according to the authors, unreliable at a large scale, presenting high rates of false positives or requiring impractically long flow observations.

This state-of-the-art attack consisted of training a convolutional neural network (CNN) to learn the intrinsic signature of traffic in the Tor network – the network's correlation function – before attempting to correlate specific flows. In fact, the complex nature of noise in Tor and the unpredictability of perturbations in the network were pointed to as reasons for the existing correlation attacks' inefficiency in large-scale, real-world scenarios.

The system used a CNN because this type of neural network is known for having good performance on time series and the previously mentioned correlation function of Tor could be modeled as one. The CNN is composed of two layers of convolution and three fully connected layers. It can be seen as a pipeline where flow pairs are given as input and a value between 0 and 1 is returned as output. Each flow is represented by four vectors – two for upstream and downstream inter-packet delays and two for upstream and downstream packet sizes – and the flow pair is represented by a eight row matrix constructed from both flows' vectors. The output represents the probability of the two flows being correlated. The first convolution layer is intended to capture the similarities between adjacent rows of the previously mentioned eight row matrix while the second has the goal of capturing overall traffic features from the combination of timing and size information.

The CNN needs to be put through a training phase in order to "learn" to recognize correlated flows and, to this extent, the authors presented it with two large sets of flow pairs, one where the pairs were correlated and the other where they were not. The pairs of each set were labeled with the values 1 and 0, respectively. During testing the CNN is presented with unlabeled flow pairs and has to output the believed probability by itself.

DeepCorr was evaluated using flows collected by accessing the top 50000 Alexa websites via Tor. Half of the collected flows were used for training purposes and the other half for testing. From each flow only the first 300 packets were used and padding was applied to shorter flows. The authors evaluated the system's performance over a period of one month and concluded that re-training would be necessary approximately every three weeks.

In comparison to other state-of-the-art systems such as Raptor [25], DeepCorr was able to achieve greater accuracy for long observations and significantly greater accuracy for shorter observations. In particular, with only 900 packets (approximately 900MB of data), the system achieved 96% accuracy while Raptor only achieved 4%. And while Raptor was able to present 96% accuracy with 100MB of data, DeepCorr was able to reach 100% accuracy with only 3MB. These accuracy values came at the expense of computation time, as DeepCorr was roughly two times slower than Raptor for same size observations. However, since DeepCorr needed significantly smaller observation times/sizes to achieve the same level of accuracy as previous systems, it ended up being much faster than the others when aiming for a specific accuracy value.

DeepCorr's very high accuracy with small observation volume required made it a staple in what passive traffic correlation attacks were concerned. The use of CNNs, the main factor for its success, was not without its disadvantages, however, as the need for periodic retraining is given as one of hindrances to its widespread, long-term use. The combinatorial nature of DeepCorr's approach for comparing flows is another of the negatives of this system as it is very expensive from a computational point of view.

Torpedo

More recently, a new attack was proposed based on a system named Torpedo [16]. Torpedo is a system developed with the goal of enabling a passive, global adversary to launch AS-level deanonymization attacks on Tor onion sessions. It relies on the attacker's ability to observe and log traffic information at each of the controlled ASes to launch correlation attacks based on deep learning. The Torpedo system works on a single pipeline where flows are gathered, filtered, classified and correlated in a total of six stages divided into two phases: *filtering* and *matching*.

When collecting the network traces that will become the basis of the attack, Torpedo is careful to do a selection based on information regarding the Tor guard relays available at the moment of sampling. This allows it to avoid logging flows unrelated to Tor circuits and acts as a first filter for the system. Once the collected flows enter the Torpedo pipeline itself they begin the filtering phase on the *origin checker*. This module leverages the differences between Tor client and Tor OS traffic [33, 34] to classify the flows according to their origin using machine learning, doing so with great accuracy.

The second stage of the filtering phase is the *request separator*. This module attempts to isolate individual page requests from clients as well as OS responses. It does so by adopting a sliding window approach whereby bursts in the captured network traces are grouped into bins according to the amount of data sent in a given time slot. Consecutive bins where the amount of data captured is above a certain threshold are grouped in requests, with a few nuances to account for eventual flow delays imposed by the network. This stage, compared to the previous one, is not as successful. In fact, Torpedo's evaluation shows a significant decline in recall percentage (from 14% to 1.3%) when introducing the request separator against the ground truth, as it was only able to successfully separate any requests from 25% of the tested sessions and in only 2% did it do so for all of the session's requests.

Once the requests are separated, Torpedo runs the *OS request identifier* on the requests originating at the clients. The goal of this stage is to use machine learning classifiers akin to the ones form the origin checker to determine which requests are OS-bound or not. This is possible since a distinct bursting behaviour was identified when directing requests to OSes when compared to regular web pages. This stage, although not as accurate as the origin checker, proved to be moderately successful, being responsible for a slight decrease in recall percentage (from 1.3% to 1.2%) against the ground truth.

The next module is also the first in the matching phase of the pipeline and is called *ranking stage*. The ranking stage receives as input the individual OS-bound client requests as well as the OSes' responses and ranks each possible flow pair according to the confidence that both flows are correlated. This ranking is meant to serve a coarser and faster filtration of flow pairs, allowing for the reduction of the search space before the more sophisticated, slower, correlation stage takes place. The confidence

scores, form 0 to 1, are determined through the use of a neural network with three fully connected layers, and is based on *packets in forward direction*, *packets in reverse direction* and *time of capture*.

Once a coarse selection of flow pairs is done, the most likely correlated pairs move on to the *correlation stage* of the pipeline. This stage is based on the architecture of DeepCorr [14], with slight modifications. It makes use of a convolutional neural network, composed of two layers of convolution and pooling and three fully connected layers, that takes as input the time-series of inter-packet timings and packet sizes of each flow pair. This network is trained with two sets of flow pairs, one whose pairs are correlated and another whose pairs are not. The output of this stage is a score from 0 to 1 representing the probability of a client-generated and a OS-generated flows being correlated.

The final step is called *session analysis stage*. Flow pairs are grouped according to specific timing characteristics – namely, similar timing differences between the first packet sent by the client and the first packet received by the OS – and assessed on whether the resulting bucket is an adequate reconstruction of a client-OS session. Candidate sessions are derived according to a minimum number of estimated requests per session and scored. The score is calculated from the average scores of all requests inside that session's largest bucket and compared to a threshold in order to deem the session as successfully correlated.

Despite attaining high precision, Torpedo suffers from both low coverage and success rates. This is likely due to Torpedo's filtering and correlation steps being very expensive from a computation standpoint and poorly scalable, especially when considering the possible $N \times M$ relation of N users to M OSes at any given time. With the system we propose we intend to tackle these issues by leveraging an active approach to narrow down the problem scope. Our system will be able to reliably identify the OS's identity which would make Torpedo's usage more practical.

2.3 Watermarking Techniques

In this section we cover the topic of watermarking techniques. The term watermarking was used in the context of covert channels by Wang et al. [35] in 2001 where a watermark was defined as "a small piece of information that can be used to uniquely identify a connection". Network flow watermarking is, therefore, a type of traffic analysis where a watermark is embedded into flows, near their source, allowing for their identification at a specific observation point of the network [36]. For the remainder of this report we will consider that a watermark may be multi-bit, although we are aware that some of the literature refers to *watermarks* as being one bit and *fingerprints* as multi-bit.

Flow watermarking techniques may be, and are in fact, used for many different purposes ranging from thwarting network cyberattacks [37] to launching them [32]. We are mainly interested in the latter but, before mentioning a few particular attacks resorting to this approach, we begin by briefly describing the flow watermarking process as a whole.

The technique of traffic analysis with flow watermarking may be divided in two phases: converting information into a watermark, and embedding it into the flow – carried out by a system component called *watermarker* – and observing a flow, being able to identify it as watermarked and decoding and

retrieving the information from the watermark itself – carried out by a *watermark detector*. In particular, and according to Iacovazzi et al. [36], the watermarker is in charge of: selecting which target flows will be embedded with the watermark; codifying the information into symbols which will be mapped into a sequence of bits which will, in turn, be transmitted through the watermark; selecting a *diversity scheme* for spreading the watermark bits; and actually embedding the watermark into the carrier signal.

2.3.1 Diversity schemes

Diversity schemes describe how a signal spreads in specific domains and are closely tied to the chosen carrier. There are three classes of diversity schemes: *time*, *frequency*, and *space*. Time diversity is based on replicating a given pattern over time in a recognizable way [31, 38]. This can be done, for instance, by inducing controlled delays in a flow's packets. Frequency diversity uses interference in a flow's rate to embed a pseudo-noise code that allows for the transmission of the watermark [39, 40]. Finally, space diversity uses multiple flows similarly watermarked in order to transmit a single signal. This is done so that a lightweight, otherwise difficult to detect, watermarking signal is amplified enough for detection [37].

2.3.2 Carriers

Watermarking techniques also vary in carrier. The carrier is the particular traffic feature where the watermark will be embedded. Iacovazzi et al. [36] categorize carriers into *content*, *timing*, *size* and *rate-based*. Content-based carrier watermarking resorts to embedding the watermark directly in the contents of the exchanged messages, be it in their headers or payloads [35]. Since encryption is often put in place, this type of carrier is seldomly used. Timing-based approaches, arguably the most popular, carry the signal in the sequence of arrival and/or departure times of packets measured at a given point in the network by introducing controlled delays to specific packets. The contents of the watermark may be calculated as a function of simple inter-packet delay (IPD) values and/or packet departure times [31] or through more mathematically complex operations involving the mean balance of either IPDs, interval centroids or interval packet counts [38]. Size-based techniques usually resort to altering flow packet lengths by an amount derived from the watermark value to be encoded [41]. Similarly to content-based carriers, this method also requires access to the packets before encryption is applied, which decreases its overall applicability and appeal. Finally, rate-based watermarking carriers consist of fluctuations imposed on the real traffic rate by the injection of dummy traffic in the same network segment at the same time. The amount of injected traffic can directly influence the rate at which the legitimate one is transmitted. Fluctuating the amount of "noise" traffic introduced as a function of the watermark allows for said watermark to be read from its interference on the legitimate traffic's rate [39].

2.3.3 Detection methods

The watermark detection process is relatively straightforward and depends on the encoding chosen for the watermark as well as its diversity scheme and carrier. The watermark detector begins by sniffing

the target flows from which it then extracts the relevant features according to the specific watermarking algorithm. These can be arrival timestamps in the case of time and rate-based approaches, packet lengths for size-based carriers or packet contents for the ones that are content-based. The collected features will form a descriptor vector for the observed flow that is used to compute the value of a function which will dictate whether a flow is or not watermarked or, when applicable, the actual value of more complex watermarks. Watermark decoding algorithms may be *non-blind* [31, 38] or *blind* [30] regarding, respectively, their dependence or not on carrier data provided by the watermarker.

2.3.4 Prominent Watermarking techniques

In order to present the evolution of the watermarking techniques used to attack Tor we now present some of the systems proposed in the literature throughout time in chronological order. Our objective is to not only showcase different methods used in this type of attacks but also reflect on, and learn from, the pros and cons of each one.

Wang et al. [30]

In 2003, Wang et al. [30] described a scheme for performing correlation using time-based watermarks that was deemed robust against timing perturbations. The system induced delays on specifically selected packets in order to encode a watermark into their IPDs. When performing detection, a watermark bit could be directly calculated as a function of the perturbed IPD value. The coding and decoding functions were conceived in such a way as to take into consideration a tolerable perturbation range for IPD values. Despite this tolerance, and in order to achieve a system that was probabilistically robust, the authors introduced two other strategies: repeating the watermark-bearing IPDs over a longer duration of the flow and embedding the watermark in the average of several IPDs. These redundancy strategies meant that a delay-inducing attacker could not stop the transmission of the watermark without resorting to drastic timing perturbations.

This solution's simple implementation and redundancy features made it very appealing at the time it was proposed. Today, however, it does not come as a surprise that it has become outdated, being visible and, therefore, vulnerable to more recent counter watermarking techniques [36].

Rainbow

In 2009, Houmansadr et al. proposed Rainbow [31], a watermarking scheme that would be able to use delays hundreds of times smaller than previous techniques and, as a result, be invisible to detection and robust to passive interference. Since the used delays were very small, Rainbow had to be non-blind, ie. the delay values extracted at the detector had to be compared with the delays recorded at the watermarker. The watermark was carried by and, thereafter, extracted from the packet IPDs. Watermark extraction used normalized correlation to account for network jitter and a pre-processing step aimed at rendering the system robust to packet addition and removal.

The evaluation conducted over Rainbow's prototype led the authors to conclude that the system was not only invisible to detection but also presented false error rates "orders of magnitude lower for short observations" when compared to previous solutions.

This system constituted a clear improvement from its predecessors, like Wang et al.'s [30], given its ability to not only use much shorter time perturbations in the original flows but also to achieve much lower false error rates for shorter observations. Its main drawback, other than the fact that it has since become outdated and vulnerable to countermeasures [42, 43], is the fact that it relies in non-blind watermark detection, making its implementation and execution more complex.

Swirl

In 2011, Houmansadr et al. proposed Swirl [38], a new watermarking scheme that was, unlike other existing techniques, scalable, invisible and resilient to packet losses. Their main insight was the fact that the watermark pattern was tailored to the characteristics of each flow being marked.

Similarly to Rainbow [31], Swirl watermarked flows via the introduction of very small delays and thus shared its invisibility to detection tools. How it did so, however, greatly differed from the Rainbow approach, as Swirl did not simply delay packets while recording the delay locations for sharing with the detector. Instead, its approach consisted of dividing a flow into intervals, these intervals into subintervals and yet again into slots before using timing data from a specific interval to determine which delays to apply elsewhere in the flow. Doing so meant selecting two intervals, the *base* and the *mark*, and the base interval was used to derive parameters necessary for embedding the watermark into the mark. According to a specific watermark key and the parameters derived from the base, packets from the mark interval were delayed as to be permuted to specific slots in said interval.

Detection consisted of analysing the base, extracting the necessary parameters and checking if the packets in the mark followed the imposed watermarking distribution.

This system boasted better accuracy than its predecessors under ideal conditions while still being, at the time, invisible. This is greatly due to the fact that this scheme tailored the watermark to the target, something that the previous proposals did not. However, its main drawback stems from the idea that such high accuracy was achieved under "ideal conditions" which meant the need for relatively long flows (at around 2 minutes). Moreover, and like the previously mentioned systems, Swirl has also become outdated and is now vulnerable to countermeasures [42, 43].

2.3.5 Watermarking Tor Onion Services

This section serves as continuation of our chronological analysis of watermarking systems. However, we felt it adequate to separate the following techniques from their predecessors because of their focus on attacking Tor Onion Services specifically and the clear difference in approach compared to the ones covered so far.

Inflow

In 2018, Iacovazzi et al. developed Inflow [32], a technique aimed at identifying Tor Onion Services using inverse flow watermarking. This approach followed a time-based diversity scheme and carrier. In essence, an attacker leveraged congestion mechanisms to impose OS traffic patterns from a watermarker located client-side. In particular, TCP acknowledgement packets from the client to the OS were dropped in short bursts in order to temporarily stop and resume packet transmission from the OS, creating traffic gaps. The adversary controlled, as is usually assumed, both edges of the Tor circuit and was able to detect the watermarking gaps. The watermark detector, located at the OS-side, had previous knowledge of the estimated periodicity and duration of the gaps and analysed IPDs in order to identify and extract the specific watermark from the flow.

Several tests were performed on Inflow to evaluate not only the system's ability to detect *a* watermarked flow but also its performance when dealing with *incorrectly watermarked* ones. After appropriate parameter tuning, the system was able to achieve 96% true positive and 0% false positive rates. The authors also tested Inflow's robustness against known counter flow watermarking techniques (the ones described in [36]) and deemed the system robust against such attacks.

Inflow stands out from the other systems proposed at the time due to its very high accuracy and robustness against counter watermarking techniques. Its main disadvantages were, similarly to Swirl, the need for large flow lengths to reach the "ideal conditions" necessary to achieve such accuracy (3 minutes) and its vulnerability to high packet loss. The system was also more complex to implement and operate, imposing a greater computational overhead as well.

Duster

In 2019, Iacovazzi et al. introduced Duster [44], a traffic analysis attack based on watermarking directed at deanonymizing Tor Onion Services. As referred in the paper, Duster's novelty was in part due to the fact that i) it exploited Tor's congestion protocol mechanism ii) it was hidden from the target endpoint and iii) did not affect network performance.

Tor's congestion mechanism relies on a specific type of control message called *SENDME cells*. When two endpoints communicate, for instance a client and an OS, the receiving endpoint will send a SENDME cell for every 50 data cells received in a stream and another every 100 data cells received in a circuit. In the case where two endpoints communicate in a circuit containing a single stream this results in a 1-2-1-2 pattern of SENDME cells being transmitted from the receiver of the data stream towards the sender. On the other hand, the endpoint sending the data can only transmit up to 500 unacknowledged cells per stream or 1000 per circuit. Duster leveraged this mechanism and watermarked a flow by sending a large burst of SENDME cells towards the Onion Service and detecting it in the connection between the target and its guard.

In Duster's approach, a watermark detector located OS-side listened for the SENDME cell batch and consequent silence and was able to identify the flow as watermarked. In order for the OS not to notice the watermarking behaviour, the detector was in charge of keeping track of the data cells sent by the OS

and forwarding the clients SENDME cells according to the normal behavioural pattern. This essentially deleted the watermark and kept the target OS “in the dark” regarding the attack. With good parameter tuning, the authors found that the true positive rate(TPR) could be as high as 98% with false positive rate(FPR) of 3%, during evaluation. It was also determined that FPR could be lowered to 1% at the expense of lowering TPR to 94%. In early 2020, Tor set version 1 SENDME cells as the consensus default. This version introduced authenticated SENDME cells, which prevents this attack altogether since the OS could now see the SENDME cells being sent by the detector instead of the guard node.

While Duster was able to achieve extremely high accuracy and invisibility, the fact that Tor changed the SENDME cell default to require authentication has made it unusable.

Finn

In 2021, Rezaei et al. introduced Finn [15], a blind watermarking technique based on deep learning and inspired by DeepCorr [14]. As many other systems do, Finn uses a time-based diversity scheme and carrier. In particular, and again similarly to other solutions, Finn’s watermarks get embedded via delaying packets in the target flows. The main insight introduced in this particular system is that is leverages neural networks to avoid a “manual” process for embedding and extracting the watermarks.

Finn’s watermarker is made up of a fully connected network of four hidden layers. As input, it takes the watermark to be embedded (a vector of all zeros and a single one) and the network noise at the target flow. The output is a vector of delays to be added to the flow’s packets and whose size is equal to the total number of packets in the flow. The generated delays take into account the network noise expected to be introduced to the flow such as to prevent it from damaging the watermark. Therefore the network noise is also added to the flow.

The decoder consists of a network of two convolution layers and two fully connected ones. As input, it takes the noisy and watermarked IPDs extracted from the received flow. The two convolution layers extract the encoded noise as well as the one that might be naturally added by the network. The fully connected layers extract the watermark itself by returning a vector of probabilities with the same size as the number of bits in the watermark. Each value represents the probability of the respective bit being a 1. Since the watermark is made up of only zeros and a single one, the highest probability is taken as the one’s “location”.

The model is trained with data regarding different IPDs, watermarks, watermark delays and network noise. The evaluation conducted on the live network allowed the authors to determine that, after parameter tuning, the system was able to achieve TPR between 93% and 98% and fixed FPR of 0% for single bit watermarking over different types of connections. A comparison with DeepCorr shows that Finn achieves sightly better results but with much less data (50 packets comparing to DeepCorr’s 300). Evaluation also showed that Finn is robust against real-time noise not used for training and extremely difficult to detect.

Factors such as Finn’s high accuracy, small data volume requirements, high robustness and low detectability make it one of the most capable systems the state-of-the-art has to offer in the field of watermarking-based traffic correlation attacks. This solution does not come without its own set of draw-

System	Diversity Scheme	Carrier	Blindness
Wang et al. [30]	Time	Time-based	Blind
Rainbow [31]	Time	Time-based	Non-blind
Swirl [38]	Time	Time-based	Non-blind
Inflow [32]	Time	Time-based	Non-blind
Duster [44]	Frequency	Rate-based	Blind
Finn [15]	Time	Time-based	Blind
DissecTor	**Frequency/Space**	**Rate-based**	**Non-blind**

Table 2.1: How DissecTor compares to the studied watermarking systems.

backs, however, and we can mention network jitter fluctuations as one of them. In fact, and similarly to DeepCorr, the neural networks that make it so powerful require regular retraining in order to cope with variations in the baseline jitter of the network. If not accounted for, such fluctuations gradually compromise Finn's accuracy, as was the case with DeepCorr. If network network conditions change frequently enough this might make the system impractical to use.

2.4 Summary

This chapter presented the relevant background information and related work on which our proposal is based. We began by introducing the Tor network and its Onion Service infrastructure and how it is used to maintain the anonymity of two communicating endpoints. Said explanation was followed by the introduction of the concept of traffic correlation attacks of which several examples were presented. This section ended by diving deeper into watermarking attacks, a subset of traffic correlation of which the system we propose is part of. We saw how these attacks are classified and presented several relevant examples. We analysed said examples and extracted both positives and negatives from each one. We conclude that this type of attacks, predominantly timing-based, can achieve extremely high accuracy and that the state-of-the-art systems do so resorting to neural network aided approaches. These techniques, however, have the disadvantage of requiring frequent retraining of the neural networks which can become a hindrance to their use. Other successful techniques that did not resort to neural networks were mainly limited by the fact that they required a large attack window, of usually several minutes, to achieve ideal conditions, which increases their visibility towards the target.

The next chapter will introduce our system, DissecTor, which will not be timing-based but will combine frequency and space based watermarking approaches. DissecTor will not rely on neural networks but will employ machine learning methods that, although requiring training, do so exclusively at attack-time which means there will be less maintenance required between uses. Finally, and seeking invisibility, our system relies on short attack windows and does not interfere with individual packets – all message exchanged with the target OS will be the same as any normal client. Table 2.1 shows how DissecTor can be classified in relation to its predecessors.

Chapter 3

DissecTor

This chapter presents DissecTor, a system whose goal is to perform the deanonymization of Tor Onion Services using a combination of watermarking and machine learning techniques. We begin by introducing the system's components and providing an overview of DissecTor's processing pipeline (Section 3.1). In Sections 3.2 and 3.3 we will be going over the chosen watermarking scheme and present our watermark amplification method, respectively. In Sections 3.4 and 3.5 we will present the two machine learning techniques used for watermark detection. The first will be focused on a supervised learning approach while the second will employ anomaly detection techniques. This chapter will conclude with the relevant implementation details (Section 3.6).

3.1 System Overview

Given its strong anonymity guarantees, the Tor network has become a staple in facilitating individuals to conduct their online activities without being identified. Whether these individuals fear prosecution from authoritarian regimes, intend to take part in illicit activities or simply seek an extra layer of security when browsing the Internet, they can all resort to Tor and its Onion Service infrastructure to access online services both as a client and a provider.

The appeal of breaking Tor's anonymity guarantees has grown since its inception as a result of its increasing popularity. In particular, the desire to identify the providers of certain Onion Services is shared by a number of entities, ranging from law enforcement agencies (LEAs) looking to put a stop to illegal activities, to overbearing authoritarian regimes seeking to silence political adversaries and activists.

DissecTor, illustrated in Figure 3.1, is a distributed system conceived with LEAs in mind. Its objective is to allow digital crime investigators to target specific Onion Services (OSes) and reveal their location in the form of IP addresses. For DissecTor to work, it is necessary to have access to a large number of network vantage points where traffic data can be collected. DissecTor assumes these vantage points are located at the Autonomous System (AS) level which can be accessed with the cooperation of several Internet Service Providers (ISPs) and/or international LEA cooperation agreements. This access is materialized in the form of modules, named *collectors*, that are to be installed throughout the network

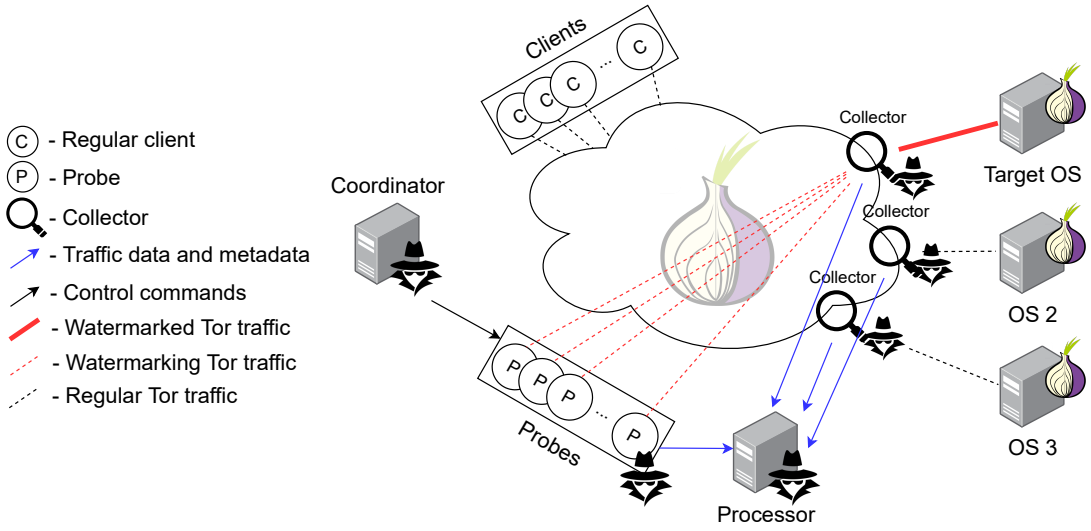

Figure 3.1: DissecTor's architecture and main component interactions

and whose job is to collect and transmit traffic data from which the target's identity can be derived. The purpose of these components, and the reason there should be a large network of them, is to collect traffic data between the OS and its guard node in the Tor circuit being used. If the network of collectors is not able to intercept said traffic, identification of that particular OS will not be possible.

Other than the already mentioned collectors, the system is made out of three more types of components: a *coordinator*; *probes* and *processor(s)*. Figure 3.1 shows how these components interact with each other and with the Tor network to perform an attack. In particular, the figure shows how the coordinator is responsible for issuing commands. It does so to both the collectors and the probes even though the arrows connecting the coordinator to the collectors were omitted from the figure as to not hinder its clarity. This component has no direct interaction with the Tor network or the target. We can also see that a varying number of probes is involved in the attack. This number is determined by the user and will directly influence the watermark signature. The probes interact with the network as if they were regular clients. The only difference between a probe and a regular client is related to the timing and number of requests sent towards the OS, as will become clear further in this chapter. Furthermore, it is also shown how the probes (each one separately) relay traffic information to the processor, as do the collectors. The processor, similarly to the coordinator, does not interact with the Tor network or the target and simply receives traffic information from the probes and the collectors. The remainder of its functionality is performed offline. The processor is responsible for watermark detection and for returning the attack's result. It is possible to distribute the processing workload over several instances of the processor component although said scenario is not represented in the figure.

For an attack to take place, all components, with exception of the probes (which are dynamically launched), should be up and running. Once the setup is complete, investigators are able to launch an attack on a specific target OS via the system's user interface. Figure 3.2 depicts an overview of DissecTor's pipeline. To best understand it let us examine the example of a law enforcement agent that seeks to uncover the IP address of an OS that serves as an online narcotics marketplace. To begin

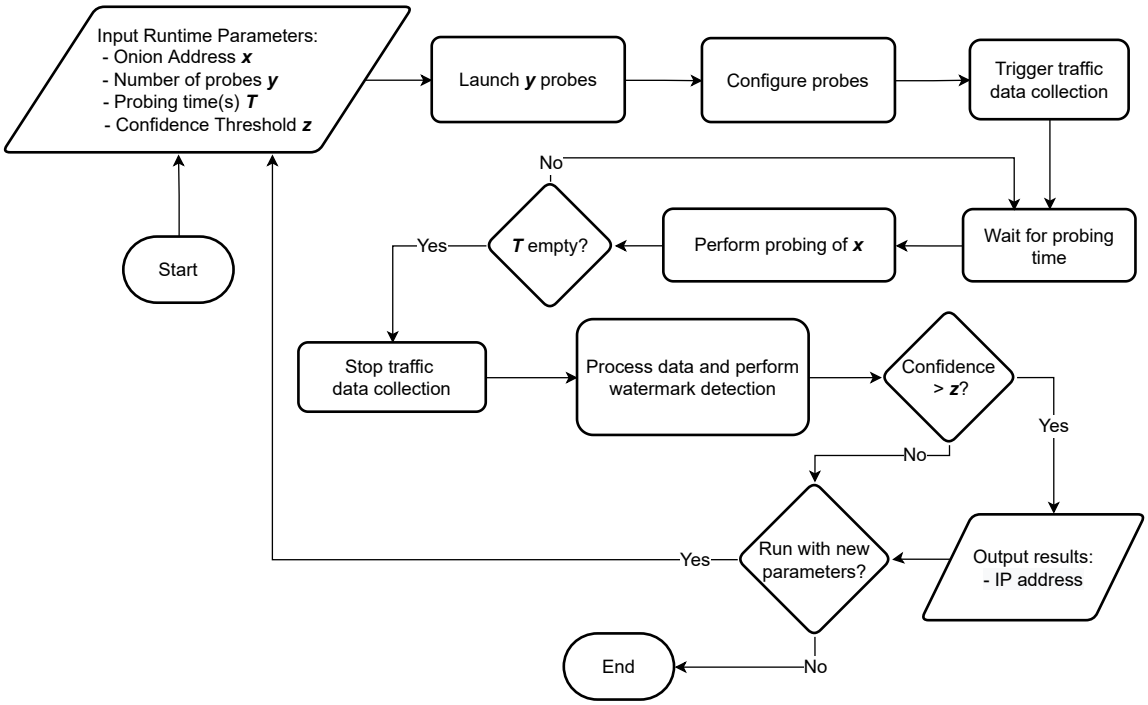

Figure 3.2: DissecTor's pipeline

the attack, the agent must input the desired runtime parameters. These consist of: the onion address of the target x; the number y of probes to use during the attack which may or may not be based of prior knowledge about the popularity of the OS; a set of probing times T which must have one or more timestamps of when the probing sessions are to take place and, finally, a confidence threshold z. In the example of an attack on the narcotics marketplace, the agent might input *verycheapdrugs.onion* as variable x, 10 as y, 3 different times the next day as T (11:00am, 1:00pm and 3:30pm) and 80% as z.

Once the runtime parameters are collected, the coordinator, which is responsible for the synchronization of all other system components and the overall orchestration of the attack, launches and then configures 10 probe instances. Some amount of time before the first probing session, at 11am, the coordinator signals all collectors to start recording traffic data. The amount of time between the beginning of the collection and the first probing session is dependant on the watermark detection technique in use.

At 11am, when the first probing session is to take place, all 10 probes concurrently send requests to *verycheapdrugs.onion* such that a traffic pattern, hereby referred to as a *watermark*, is directly embedded in the ingress and indirectly embedded in the egress traffic connecting it to its guard node. During the session all probes are responsible for collecting metadata, namely timestamps and volume of all network layer packets, which will later be used to guide the watermark detection. Once the 11am session in concluded, the system stays idle until the next probing time, 1pm, arrives and the process is repeated.

Once all probing sessions take place the coordinator signals the collectors to stop recording traffic data. The data, both from the collectors and the probes, is sent to the processor which is responsible for conducting the last phase of the attack, the *watermark detection*. To do so, it filters, conducts statistical analysis on, and then applies machine learning techniques to the collected data. If the system identifies one or more IP addresses whose likelihood of belonging to *verycheapdrugs.onion* exceeds 80% these

are then presented to the investigator. In case the system finds no suitable IP addresses this might mean several different things: the confidence threshold was too high; the number of probes was insufficient either for the baseline popularity of the OS or for the volume of traffic sensed at the time chosen to perform the probing sessions; or the connection between the target and its guard is not covered by the network of collectors. In any case, the investigator may want to run the attack once more with different parameters or give up the pursuit for the meantime.

The main challenge when conceiving the system consisted of developing the watermark itself. Specifically, a balance had to be achieved between a watermark that is reliably detectable by the system while being covert enough not to be detected by the target. Simultaneously, the watermark should be resistant to typical traffic fluctuations. The next section will present the watermarking scheme used in DissecTor.

3.2 Acceleration-Based Watermark

As was mentioned in the section above, DissecTor employs a new watermarking technique – named *acceleration-based watermark* – based on induced traffic patterns detectable on the connection between the target OS and its guard node. These patterns are manifested in the form of throughput spikes visible in both the ingress and egress flows of said connection. For this reason we can classify our watermark as taking advantage of a rate-based carrier. Its diversity scheme can be seen as combination of frequency and space as will soon become clear. Finally, the watermark detection is non-blind, relying on data provided by the watermarker (which in our system consists of the set of probes).

To embed the watermark, each probe, i.e., the watermarker, acting as a regular client, fetches the target's landing page while strictly registering the departure and arrival times of both the request's and response's network packets. This request, which is not different from any other the OS might receive, is directly reflected on traffic flowing between the guard and the OS in the form of an increase in the data-rate followed by its decrease to previously existing levels. This behaviour, induced by every request sent towards the OS, can be seen as a small spike which can be located within the time window delimited by the departure time of the request and the arrival time of the response recorded by the probe. To be more precise, if we separate traffic according to its direction we find there are two distinct spikes, respectively in the ingress and egress flows, the former being induced by the request and the latter by the response. These spikes differ from one another in both amplitude and the time frame they take place in and can both be used in conjunction when performing watermark detection.

To perform the detection itself, the processor analyzes traffic data provided by the collectors. It begins by dividing the data according to the number of possible *marks*. A mark, identified by its IP address, is a possible match for the target OS. The number of *streams* (defined next) associated to each mark is dependant on the watermark detection method used further down the pipeline and ranges from 1 to 3. Each stream is a collection of packet sizes and timestamps which the processor then groups into time intervals of granularity G, in seconds, for easier processing. Every mark's streams are divided into the same time intervals as to allow for direct comparison between them. DissecTor may use 3 possible streams: the one containing packets sent to the mark, named *fetch stream*; the one containing packets

sent by the mark, named *reply stream* and one that contains all packets destined to and originating at the mark, named *joint stream* since it is nothing more than a combination of the previous two.

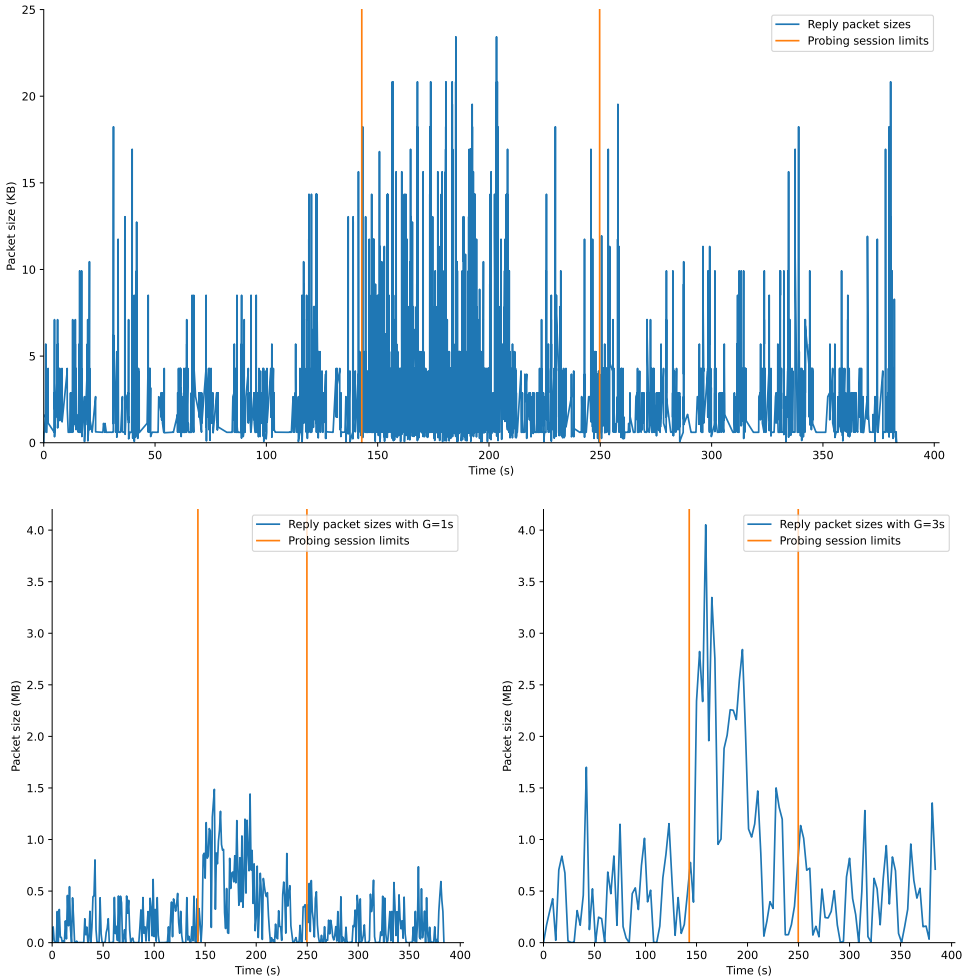

Figure 3.3: Top: Size of network packets recorded from the reply stream of a target OS. Bottom Left: Grouping the recorded packet sizes with G=1s. Bottom Right: Grouping them with G=3s.

Comparing each mark's streams is the next logical step in the process. Since every request received by the OS is responsible for its own pair of throughput spikes, our goal is to detect DissecTor's spikes (the watermark) from the set of candidates. The need to compare spikes, which consist in fluctuations of throughput, given in B/s, led us to choose the *acceleration*, given in B/s^2, as the natural metric to use in our analysis. Using the acceleration is also adequate when comparing sets of different candidate flows whose baseline/average throughput vary widely since the induced variation is independent of those and only relates to the OS's response size, time and the number of requests sent by the probes. Figure 3.3 shows how the same set of packet size samples might be grouped according to different granularity values G. This data can then be used to calculate the throughput in a given interval Δt (of size G), which is given by Equation 3.1 where Δs is the total amount of bytes transmitted during said interval.

$$throughput = \frac{\Delta s}{\Delta t} \tag{3.1}$$

Figure 3.4 shows the result of calculating the throughput of the stream from Figure 3.3 for two different

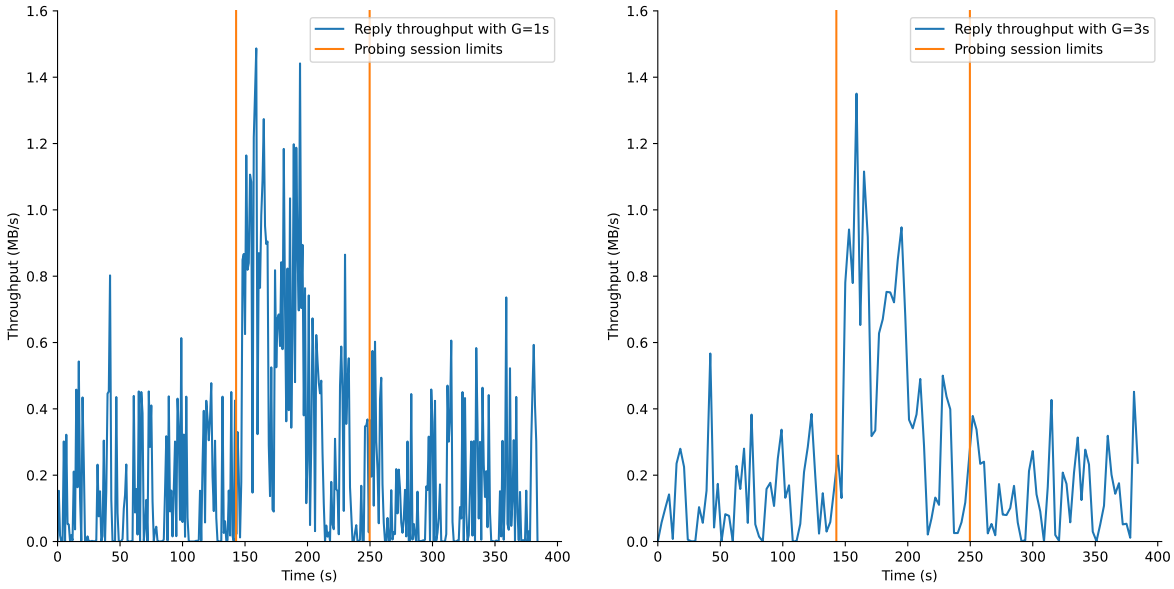

Figure 3.4: Stream throughputs derived from the packet size data from Figure 3.3. On the left G=1s and on the right G=3s. A cleaner signal can be observed with a coarser time interval (G=3s).

values of granularity G. These throughput values can then be used to calculate the acceleration between consecutive samples. The acceleration in a given interval Δt is calculated according to Equation 3.2 where $\Delta throughput$ is the variation of throughput during Δt.

$$acceleration = \frac{\Delta throughput}{\Delta t} \tag{3.2}$$

After calculating a vector of accelerations, where each value corresponds to an interval of size *G* for a given flow, these are grouped into *buckets*. The size of each bucket depends on the duration of the probing sessions and is calculated according to the metadata collected by the probes. Specifically, every acceleration value corresponding to packets observed during a probing session is grouped into the same bucket. For example, in the case of Figure 3.5, we could determine that a single bucket would be formed between the two vertical orange lines, which delineate the time when a probing session took place. This means that if a given stream contains data pertaining to two different probing sessions, two separate and possibly different-sized buckets will be formed in accordance to the probing sessions' durations. The remaining acceleration values in the vector do not correspond to the watermark and are grouped into buckets either by duration *B* (as many B seconds sized buckets as possible) or by divisor *D* (grouped into D equally sized buckets). Although there are several ways of forming the buckets, once a method is chosen it must be applied to all marks' acceleration vectors. This ensures that the n^{th} bucket of every stream starts and ends exactly at the same timestamp and can, therefore, be compared.

As the next step after forming buckets of acceleration values and labeling them as *"watermark candidate"* or *"normal traffic"*, we need to compare them. To this end, a set of summary statistics is calculated for each bucket and a dataset is compiled having said statistics as features. It is important to note that the used statistics do not depend on the amount of acceleration samples per bucket and are, consequently, agnostic to their size, ensuring that the "watermark candidate" buckets do not stand out from

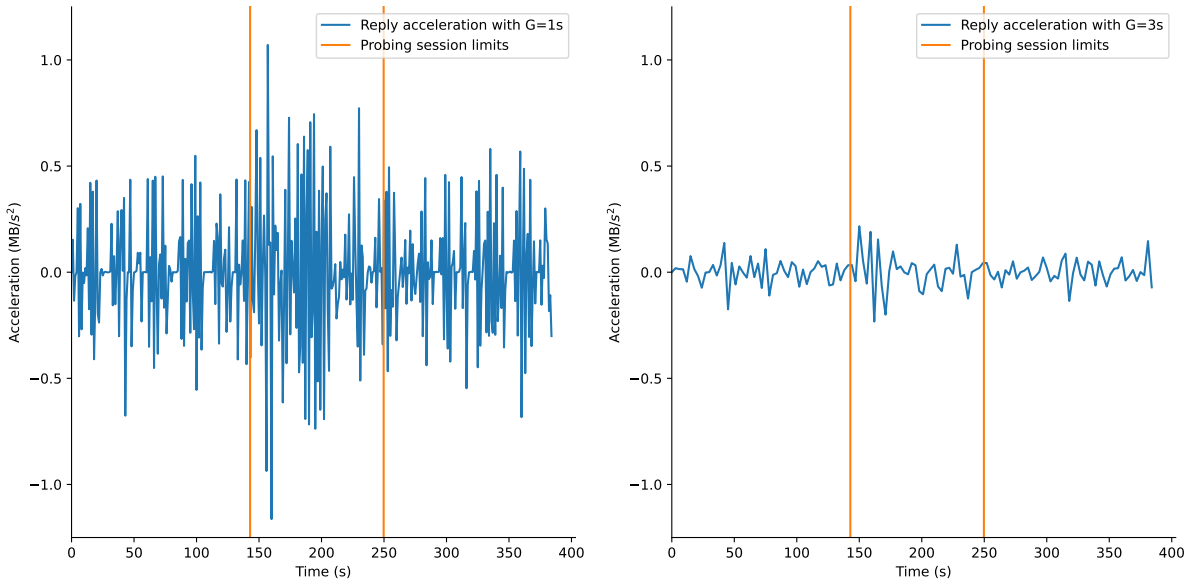

Figure 3.5: Stream accelerations derived from the throughputs on Figure 3.4. Once again, on the left G=1s and on the right G=3s

the others solely due to their duration. Lastly, after constructing datasets for every mark, the watermark detection may take place. This last step is carried out by employing machine learning techniques which will be described later on after presenting our watermark amplification technique.

3.3 Watermark Amplification

Because at any given time there might a large number of concurrent clients browsing the target OS, and since every request is responsible for its own variation in transmission rate the necessity arises to make DissecTor's variation, to which we've been referring as a spike, more identifiable. This should be done without raising suspicion from the OS.

The chosen approach was to make the probe module act as if it were a normal client sending a fetch request to the OS's landing page but have several different probes do so simultaneously in order to amplify the strength of our watermark. This is the reason our watermarking scheme may be referred to as space-based. Probe coordination is the responsibility of the coordinator, which is in charge of creating and configuring the probe instances and distributing the probing times accordingly.

This approach presents its own set of challenges since an insufficient number of probes would mean the watermark remained undetectable, being lost amidst the OS's expected baseline traffic rate but an excessively large set of probes could make the attack too obvious. The number of probes should then be chosen having an estimate number of concurrent regular clients in mind. Because such an estimate might be difficult to obtain, a possible alternative could be the adoption of an iteration-based approach in which the system performs the whole process repeatedly, gradually increasing the number of probes used in the watermarking stage until a satisfactory result, either positive or negative, can be confidently derived.

A different challenge is posed by the fact that each probe, being independent, is expected to establish its own Tor circuit to the target OS. This results in different Round-trip Times (RTT) for each probe and a significant practical difficulty in ensuring that different probes' requests will arrive at the OS simultaneously. Two possible ways of dealing with this problem are i) to increase the number of successive requests each probe sends in a probing session and ii) to decrease the granularity used by the processor when calculating the accelerations. The former increases the likelihood of the requests overlapping and the latter allows the impact of several non-simultaneous but closely timed requests to be grouped in the same calculation. Nonetheless, both approaches have their disadvantages as the first implies longer, less precise and less covert attacks and the second less data points and consequent contamination of watermark data with baseline traffic.

Now that we explained how DissecTor performs the watermarking step of the attack, we will present how the watermark detection stage is carried out. The next section introduces the first of two methods we studied – using supervised classifiers.

3.4 Watermark Classifiers – Training and Detection

Once the processor has formed the acceleration buckets for every mark's streams it employs machine learning techniques to analyze them in order to determine which stream, if any, contains the watermark. The first approach we explored consisted in the use of classifiers to detect the watermarked buckets.

In order to employ this methodology we first need to train a model, providing it with data points of both watermarked and non-watermarked buckets. These data points are described by the set of summary statistics previously mentioned which will constitute the features for the datasets used by the machine learning algorithms. Each bucket's class (watermarked or not) is determined by whether it contains timestamps where a watermarking session took place. The set of summary statistics we used were the following: maximum, minimum, variance, skewness, kurtosis, geometric mean, harmonic mean, mode (and mode count), moment, nth k-statistic for n ranging from 1 to 4, nth k-statistic variance for n ranging from 1 to 2, geometric standard deviation, interquartile range, diference between the 5th and the 95th percentile, standard error of the mean, bayesian confidence intervals (for mean, variance and standard deviation), differential entropy, median and median absolute deviation. This statistical approach to feature selection is similar the one employed in the encrypted traffic fingerprinting literature [45–48]. Once the model is adequately trained we provide it with the buckets marked as "watermark candidate" which the model then classifies as watermarked or not. Since several classification models exist it is essential to choose the best one for the task at hand. We compared several classifiers according to their True Positive (TP) and False Positive (FP) rates. These results will be provided in the following chapter.

The biggest drawback of this approach resides in the necessity for training the classifier. This can be specially difficult since we do not have access to the target OS's traffic data before-hand. In order to use this method a mock scenario must be set up where the expected conditions at the time of the attack are simulated. This implies having prior knowledge about the target, such as approximate number of concurrent users, in order to select the number of probes to use, and an estimate baseline ingress

and egress throughput to be artificially recreated. It also implies setting up a replica OS, applying the determined conditions and conducting an attack. The datasets that result from this attack are the ones used to train the model. All this should be done before the real attack takes place.

Given that, as previously mentioned, much of the required information is difficult to attain, this approach poses obvious obstacles to investigators looking to employ it. A different approach, that does not rely on privileged information, is therefore much more suited to this application. The next section presents such an technique and how we used it to conduct the attack.

3.5 Anomaly Detection

We conducted a set of experiments that rely on anomaly detection techniques instead of supervised classifiers to identify the watermarked buckets by their statistical differences from unwatermarked ones. The methodology we employed is similar to Zhang et al.'s [49] and relies on One Class Support Vector Machines (OCSVMs) [50] as an adaptation of Support Vector Machines (SVMs) to one class classification problems as is the case of determining the presence or absence of our watermark.

Contrary to the previously covered classifiers, where a model is trained with samples from all classes, using OCSVMs only requires data points belonging to one of the classes. As Zhang et al. [49] explained, this data is then mapped into a multidimensional space based on its features after which a decision function is derived. This function is afterwards used to determine if new data points belong to the class used in the training stage or if they are anomalies, therefore belonging to the *other* class.

For our specific application, this methodology eliminates the need to construct mock scenarios in order to collect samples of *both* watermarked and unwatermarked buckets. In fact, using OCSVMs for anomaly detection requires only unwatermarked traffic as training for the model. Since this traffic collection process can be done immediately before and/or after a probing session takes place, we can ensure the OCSVM is trained with the most realistic data by simply adjusting when to start and stop the collectors at the time of the attack. Furthermore, this technique makes it more practical to have a OCSVM for each mark, ensuring the greatest possible fit to each candidate instead of employing a *one fits all* approach.

To clarify, in order to use this technique in DissecTor we must ensure to collect enough unwatermarked samples before and/or after the probing sessions. The buckets may be formed in exactly the same way. Instead of having a single previously trained model reliant on data points collected in mock scenarios, there is a OCSVM for each mark that is trained exclusively with the unwatermarked buckets collected from said mark's stream(s). After training each OCSVM, we present it with the buckets collected from that specific mark at the time of the probing sessions. All streams not belonging to the target will have their buckets classified as unwatermarked and the target's stream should have its buckets classified as an anomaly, effectively deanonymizing it.

The accuracy and effectiveness of this methodology is in large part reliant on the decision function determined by the OCSVM as best fitting the data. Several parameters can be fed to the algorithm in order to alter this function's behavior, specifically how the margin separating the class and the anomaly

is determined. In the next chapter we will present the results obtained when employing this technique for several combinations of these parameters.

3.6 Implementation

We implemented DissecTor in Python v3.8.10 writing about 4400 lines of code. Most of the main system components, namely the coordinator, probes, OS and clients (for testing purposes) where deployed on Google Cloud Compute Engine instances each running an Ubuntu 18.04 image.

The deployment of the necessary instances was automated via Terraform v1.1.8 and their configuration was handled through an Ansible v2.9.6 playbook ran from the coordinator. An effort was made to ensure the instances were deployed in several different geographical locations namely in Europe, North America and Southeast Asia. All nodes required to communicate via Tor (OS, clients and probes) had Tor v0.4.2.7 installed and both client and probe instances had, in addition, access to Tor Browser v11.5.1. These nodes all ran on Docker containers. The probe and OS containers had 2vCPUs and 8GB of RAM each while the client containers had 1vCPU and 4GB of RAM.

In order to perform the necessary Tor requests programmatically, both client and probe instances used a Python package called TBSelenium. This package extends the functionality of the browser automation tool Selenium to function with the Tor Browser. Other important aspects to note in regards to the requesting mechanics of the system include the fact that we configured TBSelenium to disable caching when performing the browsing sessions, opted to make the client wait for the totality of the response before sending subsequent requests and imposed a 30 seconds timeout for each one.

The functionality of the collector was simulated by capturing traffic at each communicating party's Docker container's network interface, using Tcpdump v4.9.3 and Libpcap v1.9.1 to do so and saving the data in .pcap files. To process this traffic data we used Python scripts and the Dpkt v1.9.7.2 package to filter the captured packets keeping the ones pertaining to non-empty TCP packets resulting from Tor activity and extracting size and timestamp information from them. We also separated packets according to their direction, allowing for the segregation of inbound and outbound traffic. This information was saved in .pickle files for later use. When several probes were involved in the same probing session, data contained in the aforementioned .pickle files was merged into two (one for each direction) descriptive of the session. Each OS also resulted in two .pickle files containing size and timing information for all packets that shared a direction. Attention was given to sort the data by timestamp when merging files, as a failure to do so would have had dire consequences in the system's ability to perform its task.

The next step consisted of using the data contained in the OS's .pickle files to calculate acceleration values for a certain granularity G. We used a Python script to do so and a new .pickle file containing accelerations and timestamps was created.

After this a set of Python scripts was responsible for grouping acceleration values into buckets based on probing session timing information and the chosen methodology for the formation of the "non-watermarked" buckets. Each bucket was labeled as belonging to a probing session or not. After the bucket formation process was concluded, the set of summary statistics was calculated for each bucket

using the Statistics module of the Scipy v1.8.1 Python package. Pandas v1.4.2 was used to create a dataset where each bucket corresponded to an entry. This dataset was stored as a .csv file.

We used two different methods to detect watermarks. Both relied on the same dataset constructed in the previous step. The first detection methodology, based on classifiers, was conducted with the Weka Machine Learning software [51] v3.8.6 and the second approach, utilizing One-Class Support Vector Machines, was performed with the SVM module of the Scikit-learn [52] v1.1.1 Python library.

Summary

This chapter presented DissecTor, a system aimed at performing the deanonymization of Tor Onion Services by employing an acceleration-based watermarking technique and machine learning for watermark detection.

We presented DissecTor's components, their functionality and how they come together during an attack resorting to an illustrative scenario of a LEA seeking to take down a narcotics marketplace.

We elaborated further on the watermarking technique the system employs, reflected on the balance needed between detectability and covertness and presented our approach to watermark amplification.

Furthermore, we proposed two approaches for watermark detection that make use of different machine learning techniques. The first approach resorts to classifiers while the second is based on anomaly detection. Besides presenting these techniques we reflect on their applicability.

We concluded this chapter by presenting details regarding DissecTor's implementation.

Chapter 4

Evaluation

In this chapter we present DissecTor's evaluation. We begin by describing the methodology used to evaluate the system, including the metrics used to assess the performance of DissecTor and the experimental testbed (Section 4.1). We then present the results for a benchmark execution of the attack for each of the watermark detection techniques, comparing them (Section 4.2). Finally we conclude by presenting several variants for each technique where we studied how changing different system parameters impacted overall performance (Section 4.3).

4.1 Methodology

DissecTor's main goal is to perform the deanonymization of a specific Tor Onion Service. This means that our system should be able to accurately identify said OS's IP address from a pool of candidate OSes. In addition, DissecTor should be able to carry out this task in a covert fashion, i.e. without the target OS realizing that a watermark-based deanonymization attack is taking place. Considering the watermarking technique presented in Chapter 3, we argue that DissecTor's watermark should strike a balance between reliable identification and its ability not to arise suspicion.

Having the system's two goals in mind, DissecTor's evaluation is focused on a) assessing the system's ability to reliably identify a target's stream(s) as being watermarked; b) assessing whether the system has the ability to reliably identify non-target stream(s) as not being watermarked and; c) studying several possible configurations of the system in an attempt to identify the one that allows for the best results, and; d) discussing if the configurations that would allow the system to keep a "low-profile" can realistically achieve satisfactory OS deanonymization results.

4.1.1 Experimental Testbed

The experiments we conducted required the existence of an OS that would serve as target for DissecTor. This OS was to be as realistic as possible and thus a baseline traffic signature had to be established over which our system's watermark could be applied. To this extent we setup an OS serving a web page approximately 1MB in size. Furthermore we arranged a set of 4 clients that would fetch said page, wait

for it to load in its entirety, sleep a random amount between 2 and 4 seconds and repeat the process until signalled to stop.

After the baseline was established, we conducted probing experiments where a given number of probes would fetch the target's landing page and wait for it to load. This fetching operation would occur 5 times in a row without intervals. In the case of an individual request taking longer than 30 seconds, a timeout would be triggered and the following request would commence. Once all 5 fetching requests were answered for all participating probes, these would then sleep until the next session was to take place. A total of 50 probing sessions was carried out for each set of probes.

All involved parties (probes, clients, OS and coordinator) were deployed on Google Cloud Compute Engine. We used "n1-standard-1" machines for clients and coordinator and "n1-standard-2" machines for probes and the OS. The first set of machines was equipped with 1vCPU and 4GB of RAM while second made use of 2vCPU and 8GB of RAM.

While the probing experiments took place, traffic data was collected both on the side of the OS and that of the probes. As previously mentioned, the latter was necessary for the creation of the buckets that temporally located the probing sessions and the former contained the data pertaining to the traffic being examined.

In our experiments we used 1, 2, 4, 8 and 12 probes which allowed us to collect data concerning Probe-to-Client Ratios (PCR) of, respectively, $\frac{1}{4}$, $\frac{1}{2}$, $1, 2$ and 3. In the processing stage we grouped the packets in intervals of size $G = 1s$ and the buckets that did not correspond to probing sessions were created using the "divisor" strategy for $D = 1$ i.e., we formed a single bucket in the interval between consecutive probing sessions. The process was applied to the 3 different possible streams – Reply, Fetch and Joint. In total, and counting each stream separately, we ended up with 1500 buckets, 750 corresponding to probing sessions and 750 to baseline traffic. The 750 buckets corresponding to probing sessions were equally split by the 5 PCRs, at 150 buckets each.

4.2 Watermark Detection

In this section we will evaluate DissecTor's ability to correctly identify watermarked streams for two initial configurations. These configurations, one for classifier-based detection and one for anomaly detection, will serve as benchmarks and allow us to, later on, discuss how variations in the system parameters might affect performance.

Detection metrics: In order to present and compare the performance of each of these techniques we will be calculating both the True Positive (TP) and False Positive (FP) rates achieved by the system when seeking to detect watermarked buckets for all 3 streams and for all 5 PCRs. It is important to clarify what the two metrics used throughout this chapter, TP and FP rates, entail. We begin by defining as "positive" the samples corresponding to buckets that contain the watermark and by "negative" the ones that do not. Therefore, TP rate is the metric that expresses the amount of watermarked buckets correctly classified as such by each model. On the other hand, FP rate expresses the amount of unwatermarked buckets

the system wrongly classified as being watermarked.

In addition, and as a means to compare results across experiments we will make use of two other metrics: F1-score [53] and P4 [54]. Both are compound metrics that can be calculated using the number of True and False positives (#TP and #FP) and the number of True and False negatives (#TN and #FN) of a classification problem's outcome.

$$Precision = \frac{\#TP}{\#TP + \#FP} \tag{4.1}$$

$$Recall = \frac{\#TP}{\#TP + \#FN} \tag{4.2}$$

$$F1score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{4.3}$$

The F1-score metric is based on both *Precision* (given by Equation 4.1) and *Recall* (given by Equation 4.2). The precision is indicative of the amount of "positive" classifications that were actually correct while the recall, referred to thus far as "TP rate", is indicative of the amount of positive samples actually classified as such. The F1-score is defined as the harmonic mean of precision and recall and is given by Equation 4.3. F1-score gives great importance to a classifier's behaviour regarding the "positive" samples and can, to some extent, neglect performance regarding the "negative" ones.

$$Specificity = \frac{\#TN}{\#TN + \#FP} \tag{4.4}$$

$$NPV = \frac{\#TN}{\#TN + \#FN} \tag{4.5}$$

$$P4 = \frac{4}{\frac{1}{Precision} + \frac{1}{Recall} + \frac{1}{Specificity} + \frac{1}{NPV}} \tag{4.6}$$

Because of these characteristics of the F1-score we sought a second metric that would not base itself so heavily on the "positives" and attempted to be as balanced as possible. This decision stems from the necessity of DissecTor being not only capable of correctly identifying watermarked streams but also, and as importantly, of doing so without falsely flagging innocent candidates as the target. It so happens that a new metric, P4, was recently proposed with the objective of tackling F1-score's shortcomings. Besides Precision and Recall, P4 is also based on *Specificity* (given by Equation 4.4) and *Negative Predictive Value (NPV)* (given by Equation 4.5). Specificity reflects the amount of "negative" samples that are correctly classified as such (the "negative" equivalent of Recall) while negative predictive value is indicative of the amount of correct predictions out of all "negative" classifications (the "negative" equivalent of precision). Similarly to the F1-score, P4 also consists of a harmonic mean (given by Equation 4.6), this time involving all four components – Precision, Recall, Specificity and Negative Predictive Value.

	Reply Stream				Fetch Stream				Joint Stream			
PCR	TP%	FP%	F1	P4	TP%	FP%	F1	P4	TP%	FP%	F1	P4
$\frac{1}{4}$	80	2.8	0.825	0.890	76	1.6	0.826	0.892	82	2	0.854	0.909
$\frac{1}{2}$	90	1.2	0.918	0.950	90	4	0.857	0.910	80	2.4	0.833	0.896
1	90	1.2	0.918	0.950	80	2.4	0.833	0.896	88	1.6	0.898	0.937
2	94	0.4	0.959	0.975	92	1.2	0.929	0.957	94	0.8	0.949	0.969
3	98	0.4	0.980	0.988	98	0.4	0.980	0.988	98	0.4	0.980	0.988

Table 4.1: Detection metrics when performing watermark detection with the C4.5 classifier.

4.2.1 Watermark Detection via Supervised Learning

As a benchmark for the classifier-based version of the system we chose Weka-v3.8.6's implementation of the C4.5 decision tree algorithm [55]. Like all other classifiers covered in the chapter, this classifier was run with default parameters, cross validation with 10 folds and was trained with 250 samples (buckets) of baseline traffic and 50 of watermarked traffic. We did so for every stream and PCR totalling 15 executions and extracted the TP and FP rates after each one.

Table 4.1 presents the results of our experiments with the C4.5 classifier. We can observe that the TP rate ranges from a minimum of 76% in the Fetch stream when using $\frac{1}{4}$ PCR to a maximum of 98% for a PCR of 3 independently of the stream. In turn, the FP rate ranges from 4% for the Fetch stream with $\frac{1}{2}$ PCR to 0.4% in all streams using 3 as PCR and the Reply stream when PCR equals 2. Our results also reveal different trends, where i) the TP rate increases with greater PCR values ii) the FP rate decreases with greater PCR values and iii) Reply streams offer the highest TP rates and lowest FP rates of the 3, also being the most consistent with respect to the previous tendencies. This leads us to believe they are the best streams to use in this configuration of the attack.

When analysing the values of both the F1-score and the P4 we see further support for our previous claim, as the Reply stream outperforms the others in both metrics for all values of PCR with exception of $PCR = 1$ (where it performs the worst) and $PCR = 3$ (where all streams perform the same).

We consider the overall results quite satisfactory, specially given the fact that even the lower PCRs obtained TP rates up to 90% and FP rates no greater than 4%. In Section 4.3, we present the extended results of a set of experiments performed with different classifiers towards improving DissecTor's ability to detect watermarks using supervised learning techniques. When doing so, we will make use of these F1-score and P4 results to compare the different variants of the supervised learning version of DissecTor as well as the different streams and PCR values.

4.2.2 Watermark Detection via Anomaly Detection

As an alternative to the supervised learning version of DissecTor we conducted experiments to determine if performing watermark detection with anomaly detection methodology was not only possible but effective. In particular, we experimented doing so using One Class Support Vector Machines (OCSVMs) having used the implementation provided by Scikit-learn [52] v1.1.1.

Unlike the case of supervised learning, this technique requires that a model be trained exclusively

with one class of data. The intention is for the model to learn to recognize this class as best as possible. Afterwards, when faced with unknown data points the model must determine whether they fall within the class it has learned to recognize or if they are anomalies, therefore belonging to the "other" class.

In the particular case of DissecTor, all models were made to recognize unwatermarked data and treat watermarked samples as anomalies. To this extent we only trained with unwatermarked data which meant that we ended up training the algorithms once for each parameter configuration and stream instead of 5 times (as the PCR had no impact in the training set). The result is FP rates that are independent from the amount of probes used.

When running the experiments we had to define a set of parameters that influenced the behaviour of the model and, consequently, the results obtained by DissecTor. One of such parameters was the "kernel function", which controls how the model mathematically transforms training data into a decision boundary for the class being learned. While there were several kernel functions available to us, preliminary experiments led us to conclude that Radial Basis Function, or RBF, was the best since all others either performed rather poorly or did not perform at all. For this reason, although "kernel" is one of the parameters that may be changed when working with OCSVMs, we chose not to do so, using "RBF" throughout. Other parameters, however, justified a more thorough study, constituting the basis for the variants introduced later on. Next we present a brief explanation of each of these parameters.

Gamma: Defines the weight each training point has on the resulting decision boundary [56]. If the value of "Gamma" is too large the model can incur in overfitting, reporting more anomalies than it should, and if it is too small it may become too permissive, not reporting as many anomalies as expected. Although "Gamma" may take any positive value the library makes two ways of selecting it readily available – "auto" and "scale" (the default). The first defines "Gamma" as the inverse of the number of features and the second as the inverse of the product of the number of features with their variance. These were the two values we studied for this parameter.

Nu: Allows us to control both the acceptable amount training errors and the number of support vectors derived from the training set [57]. Higher "Nu" values mean that more training data points may be misclassified in exchange for a larger number of them being used as support vectors. Smaller values of "Nu" mean less classification errors may occur with the training data but the minimum number of support vectors to use will also be lower. "Nu" is a number in the interval $[0, 1[$ and its default value is 0.5.

Tolerance: Controls the algorithm's stopping criterion by defining the minimum gain each iteration must achieve to deem the algorithm worth continue running. The smaller the value of "Tolerance" the longer the algorithm will tend to run and approximate the optimal solution. Its default value is $1\mathrm{e}{-3}$.

As a benchmark for this version of the system we chose to run the model with all default parameters. As was the case for every variant reliant on OCSVM, we opted for training and testing the model 10 times per configuration and taking the mean of TP and FP rates as the final result. Each time the process was

	Reply Stream				Fetch Stream				Joint Stream			
PCR	TP%	FP%	F1	P4	TP%	FP%	F1	P4	TP%	FP%	F1	P4
$\frac{1}{4}$	91.8	52.7	0.842	0.685	77.8	47.2	0.773	0.633	91.2	53.1	0.838	0.679
$\frac{1}{2}$	84.8	52.7	0.803	0.640	90.8	47.2	0.847	0.714	89.2	53.1	0.827	0.665
1	92	52.7	0.843	0.687	74	47.2	0.749	0.611	88.6	53.1	0.824	0.662
2	90.2	52.7	0.833	0.675	91.2	47.2	0.849	0.717	89.2	53.1	0.827	0.665
3	91.4	52.7	0.839	0.683	90.2	47.2	0.844	0.710	92	53.1	0.842	0.684

Table 4.2: Detection metrics for watermark detection with OCSVM using the default parameters.

as follows: i) of the 250 unwatermarked samples available per stream, 90% was taken at random and used for training the model; ii) the model was tested with the remaining 10% in order to determine the FP rate and iii) the model was tested with each of the 5 PCRs' 50 data points to determine the TP rate.

Table 4.2 presents the results obtained when using the OCSVM approach with default parameters for each of the 3 streams and all PCRs. Overall, the TP rates were satisfactory, ranging from 74% to 92% but the same cannot be said of the FP rates, which were high when comparing with the supervised learning benchmark results, ranging from 47.2% to 53.1%.

Besides the large FP rates we find that some of the tendencies observed with the supervised learning version do not seem to be present in this case. One of the tendencies that does not translate from one version of the system to the other (although expectedly so) consists in the fact that the FP rates do not depend on the PCRs. The second difference regards the TP rates which, although showing similar values, do not seem to follow the trend of becoming greater as the PCR increases. This might hint at the possibility of this approach allowing the system to be successful in configurations requiring less probes and, therefore, more covert.

Finally, when analysing the other two metrics, F1-score and P4, we find that these indicate the Reply and Fetch streams as the best performing. The former achieved the highest scores in $2/5$ of PCR values while the latter did so for the remaining $3/5$. This leads us to believe that, for this particular configuration of the system, the Fetch stream is marginally the best suited for our analysis.

4.3 Variants

After introducing a default parametrization of DissecTor's two watermark detection techniques, we will now present the results of experiments conducted with the aim of determining if the aforementioned results can be improved upon.

In the case of the supervised learning based version, we will present the results of experiments conducted with two alternatives to the C4.5 algorithm used in the benchmark and compare the three.

As for the OCSVM-based version we will vary the "Gamma", "Nu" and "Tolerance" parameters over several values and study the impact of each of these parameters in the performance of the system. Furthermore, we will present a variation where a pre-processing scaling step was applied to the features fed into the anomaly detection technique, a procedure which has been found to improve OCSVMs' performance in the literature [58]. Finally we will present the configurations that obtained the best results

	Reply Stream				Fetch Stream				Joint Stream			
PCR	TP%	FP%	F1	P4	TP%	FP%	F1	P4	TP%	FP%	F1	P4
$\frac{1}{4}$	92	2.4	0.902	0.939	80	2.4	0.833	0.896	84	2	0.866	0.917
$\frac{1}{2}$	94	2.4	0.913	0.946	80	2.4	0.833	0.896	94	1.6	0.931	0.958
1	90	2.4	0.891	0.933	86	2	0.878	0.924	90	1.6	0.909	0.944
2	98	2	0.942	0.965	96	1.6	0.941	0.964	98	1.6	0.951	0.970
3	98	2	0.942	0.965	98	1.2	0.961	0.976	98	1.2	0.961	0.976

Table 4.3: Detection metrics when performing watermark detection with the Naive Bayes classifier.

	Reply Stream				Fetch Stream				Joint Stream			
PCR	TP%	FP%	F1	P4	TP%	FP%	F1	P4	TP%	FP%	F1	P4
$\frac{1}{4}$	84	1.2	0.884	0.929	78	1.2	0.848	0.906	82	1.6	0.863	0.915
$\frac{1}{2}$	92	0.8	0.939	0.963	80	1.2	0.860	0.914	84	1.6	0.875	0.923
1	88	0.8	0.917	0.949	84	0.8	0.894	0.935	86	0.8	0.905	0.942
2	96	0.4	0.970	0.982	98	1.2	0.961	0.976	94	0.8	0.949	0.969
3	98	0.4	0.980	0.988	96	0.4	0.970	0.982	98	0.0	0.990	0.994

Table 4.4: Detection metrics when performing watermark detection with the Random Forest classifier.

for each of the PCR-Stream pairs according to the F1-score and P4 metrics to see if they match the preceding findings.

4.3.1 Supervised Learning Variants

We chose to study the performance of two other well-known supervised learning techniques: i) Naive Bayes [59] and ii) Random Forest [60]. Both classifiers were used with the default parameters and with cross validation set to 10 folds. The datasets used in the variant experiments were exactly the same as in the benchmark which we have already covered.

Table 4.3 presents the results of the experiments conducted with the Naive Bayes classifier for every PCR-Stream pair. We can see the TP rate ranging from 80% to 98% and the FP rate ranging from 1.2% to 2.4%. We can observe some of the same tendencies that could be seen in the benchmark, namely: i) with two exceptions for $PCR = 1$, the TP rate increases with the PCR and; ii) the FP rate decreases with the PCR. The Reply and Joint streams seem to perform similarly well for this classifier. When analysing both streams' F1-score and P4 values, however, we can conclude the Joint stream is the best of the two, outperforming the Reply stream in $4/5$ of PCRs.

Table 4.4 presents the results of the experiments conducted with the Random Forest classifier for every PCR-Stream pair. In this case the TP rates range from 78% to 98% and the FP rates from 0% to 1.6%. As was the case with the two previous classification algorithms, the TP rates tend to increase with the PCR and the FP rates tend to decrease with it.

In this particular case, the Reply stream seems to present the best performance regarding both the TP and FP rates metrics, although it is noteworthy that a 0% FP rate was achieved for one of the PCR-Stream combinations even if for the largest, least covert, amount of probes. When looking at the F1-score and P4 values shown in the table we confirm the Reply stream is indeed the best suited for this particular system variant, as it outperformed the other streams in $4/5$ PCR values.

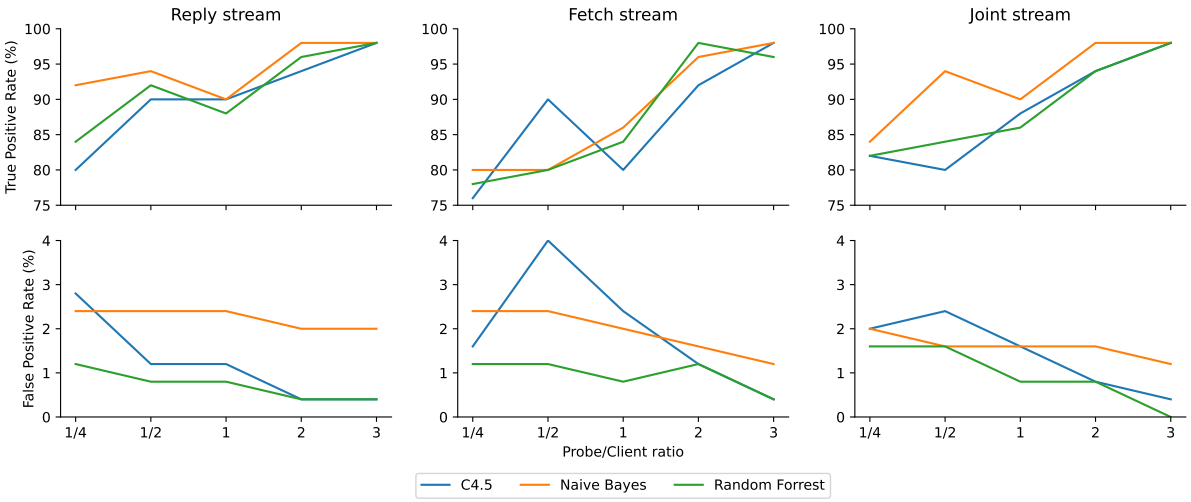

Figure 4.1: True and False positive rates for all three classifiers: C4.5, Naive Bayes and Random Forest.

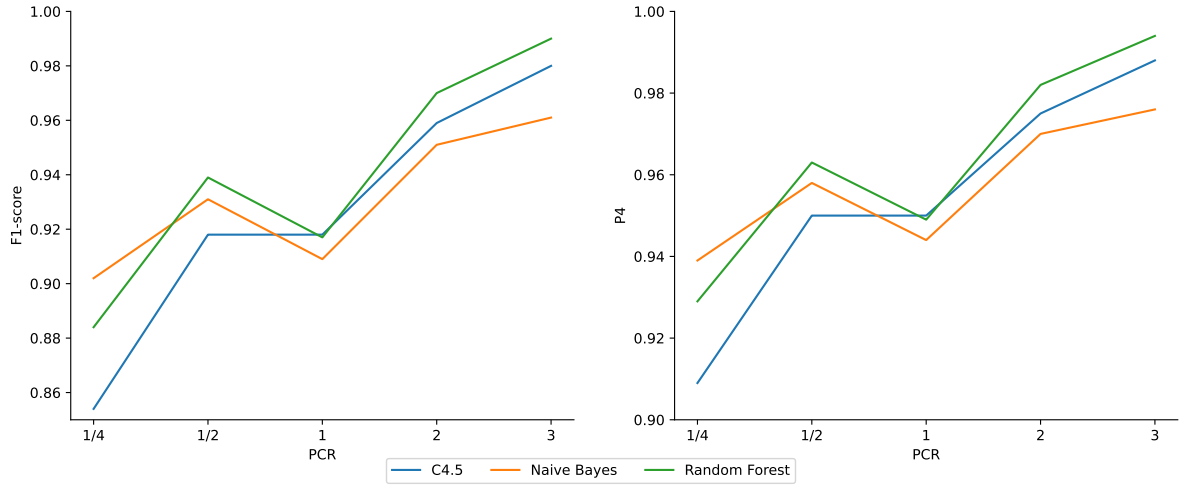

Figure 4.2: Best F1-score and P4 values of all three classifiers per PCR.

Figure 4.1 summarizes the results we discussed so far for TP and FP rate metrics, and allows for a better comparison of the classification algorithms used in the experiments. Each column of plots depicts the results pertaining to a different stream in the same left to right order as they appear in the tables. The plots on the top row present the TP rates and the ones in the bottom row present the FP rates. The color assignments are as follows: C4.5 in blue; Naive Bayes in orange and Random Forest in green.

Looking at the plots we can visually confirm the trends discussed so far as the lines in the top plots trend upwards with the PCR and the ones on the bottom trend downwards. Some exceptions to this behaviour become more evident, however. For instance, we can observe that, in the experiments conducted with a number of probes equal or slightly lower than the baseline client count, the performance of the system generally decreases but increases again once the the PCR is greater than 1. This phenomenon might be justified by the fact that, despite the watermark amplification measures allowing for better detection, the signature of the watermark is more similar to that of the baseline traffic and, therefore, more difficult to detect correctly. The reason for observing this for $PCR = \frac{1}{2}$ and not for $PCR = 2$, despite it also being adjacent to the baseline number of regular clients, is a consequence of the fact that

we did not impose any synchronization on the client's behaviour. More often then not, the number of clients requesting the OS's page will be slightly inferior to the total number of baseline clients used to determine the PCR. Therefore, while the ratio of clients actively accessing the OS to the total number of client nodes in execution will never be greater than 1 it will sometimes approach $\frac{1}{2}$, which justifies the worse system performance for this PCR value.

When comparing the performance of the different classifiers we based our analysis on the F1-score and the P4 metrics. To do so, we determined the best values achieved by each classifier for every PCR. The best value for each PCR was chosen among the three streams. Figure 4.2 presents our findings and allows us to derive some conclusions: i) there is a clear degradation in performance for $PCR = 1$, something we already mentioned but that becomes clearer to see; ii) the Random Forest classifier outperforms the others in 3/5 of the PCR values while C4.5 and Naive Bayes performed the best in 1/5 of PCRs each and; iii) although the figure does not explicitly specify it, when compiling the results it presents, we confirmed that the Reply stream is the best of the three for this analysis, followed by the Joint stream. The last conclusion is supported by the fact that the Reply stream achieved the best results in 60% of the cases, the Joint stream did so for 47% and the Fetch stream only for 13% (the total surpasses 100% due to the occurrence of some ties).

4.3.2 Anomaly Detection variants

After establishing the benchmark performance for the OCSVM-based configuration of the system for the default parameters, we conducted experiments where we tested the impact changing each one of these parameters at a time had on the TP and FP rate metrics. Besides studying the impact of varying "Gamma" (default: "scale"), "Nu" (default: 0.5) and "Tolerance" (default: $1\mathrm{e}{-3}$) we also considered whether a pre-processing scaling step would be beneficial.

The results we will present consist of the variation of each parameter separately keeping the others as their default values. This way we hope to be able to better convey their impact on the overall performance and compare them to each other in regards to their impact. In addition, we conducted experiments for every combination of parameters. Thus, this section will conclude with the "best" overall such configurations (not limited to any default parameters) and will serve as way of confirming or debunking the insights derived along the way. The "best" configurations will be determined, as has been the case so far, according to both the F1-score and the P4 metrics. Furthermore, these configurations will be used to compare the OCSVM version of the system with its supervised learning counterpart.

Standard Scaling

The datasets used so far in our analysis were directly derived from the computation of the set of summary statistics mentioned in the previous chapter. Because many machine learning techniques benefit from some form of standardization of the feature values present in the datasets, we decided to test if that was the case for this technique in particular. To do so, we used the pre-processing package available in the scikit-learn library. Specifically we utilized the "StandardScaler" class of the aforementioned package

	Reply Stream		Fetch Stream		Joint Stream	
PCR	TP%	FP%	TP%	FP%	TP%	FP%
$\frac{1}{4}$	46.4	57	39	55.7	43.6	54.7
$\frac{1}{2}$	56.8	57	34.8	55.7	54	54.7
1	52.6	57	63.6	55.7	43.4	54.7
2	55.8	57	56	55.7	58	54.7
3	47.6	57	48.2	55.7	43.8	54.7

Table 4.5: True and False positive rates when performing watermark detection with OCSVM for all default parameters after pre-processing the datasets with standard scaling.

	Reply Stream		Fetch Stream		Joint Stream	
PCR	TP%	FP%	TP%	FP%	TP%	FP%
$\frac{1}{4}$	100	100	100	100	100	100
$\frac{1}{2}$	100	100	100	100	100	100
1	100	100	100	100	100	100
2	100	100	100	100	100	100
3	100	100	100	100	100	100

Table 4.6: True and False positive rates when performing watermark detection with OCSVM using all default parameters except *Gamma* which was set to “auto” instead of “scale”.

to standardize each features’ samples x by calculating their standard score z. The formula by which z is derived consists of: $z = \frac{x-u}{s}$ where x is a sample of a given feature, u is the mean of that feature’s samples and s is their standard deviation.

Table 4.5 presents the results of performing watermark detection with the same OCSVM configuration as the benchmark (all default parameters) having pre-processed the datasets with standard scaling. We can observe that this setup results in a significant degradation of both the TP and FP rates. In fact, and unlike the previously set benchmark, the results for both metrics are very close to 50% in most PCR-Stream pairs. This performance is closer to the accuracy expected from a system classifying data points by chance and, therefore, is not suited for our application.

Gamma

The value of “Gamma” defines the influence each training point has on the decision boundary. As an alternative to the default way of calculating the value of parameter “Gamma”, we used the other standard method provided by the particular OCSVM class used throughout our experiments. Unlike the default – where “Gamma” is the inverse of the product of the number of features and their variance – the method we now analyze defines it simply as the inverse of the number of features and is designated as “auto”.

Table 4.6 presents the TP and FP rates achieved by the system for every PCR-Stream pair with the OCSVM method running with all default parameters (and no standard scaling) with the exception of the method for calculating “Gamma” which was changed from “scale” to “auto”. We can clearly see that the model classified every test data point as an anomaly, which indicates the value of “Gamma” was excessively large. In such cases, the importance given to each training set point is exaggerated and the model incurs in overfitting. Experimentally, we verified that the training points had such much influence

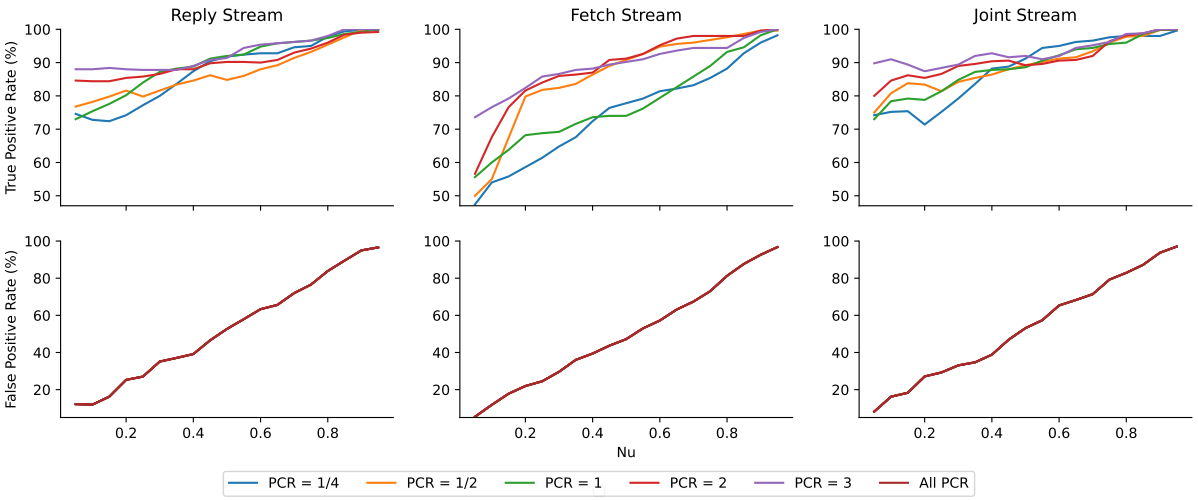

Figure 4.3: Impact of the "Nu" parameter on the true and false positive rates of DissecTor running with OCSVM in the otherwise benchmark configuration.

on the decision boundary that any point outside the training set, in this case the ones contained in the testing sets, were all considered to be outside this boundary and, therefore, classified as anomalies.

The results of our experiments suggest that this method for determining "Gamma" is not well suited for our particular application and such the default method – "scale" – should be adopted instead.

Nu

The "Nu" parameter may take any value in the interval $[0, 1[$ and controls both the tolerance regarding training errors and the amount of training points used as support vectors in the trained model. To study the impact of this parameter we opted to run the benchmark configuration for values of "Nu" ranging from 0 to 0.95 in 0.05 increments.

Figure 4.3 shows the TP and FP rates achieved by the system for each PCR-Stream pair when varying "Nu". The columns represent each of the streams in the same order used throughout this chapter. The top row of plots shows the TP rates obtained by every PCR as a function of "Nu" and the bottom row of plots depicts the FP rate as a function of "Nu". As previously mentioned, the FP rate does not depend on the number of probes, reason why there is a single brown line in the bottom plots representing "All PCRs". The color scheme for the top plots is as follows: $PCR = \frac{1}{4}$ in blue; $PCR = \frac{1}{2}$ in orange; $PCR = 1$ in green; $PCR = 2$ in red and $PCR = 3$ in purple.

From the figure, we can conclude that an increase in "Nu" results in both higher TP and FP rates although in different proportions. FP rates vary from close to 0% to close to 100% in an almost linear fashion as "Nu" increases, independently of the stream. As for the TP rates, these seem to not be affected as significantly by this parameter as the FP rates. In fact, although it is clear that TP rates increase with "Nu", the amount by which they do so does not seem to justify the drastic increase in FP. This is specially noticeable in the case of the Reply and Joint streams, where the gain from increasing "Nu" is inferior. This leads us to conclude that DissecTor may benefit from smaller values of this parameter, and that the Fetch stream may not be as good a choice as the others for this type of analysis.

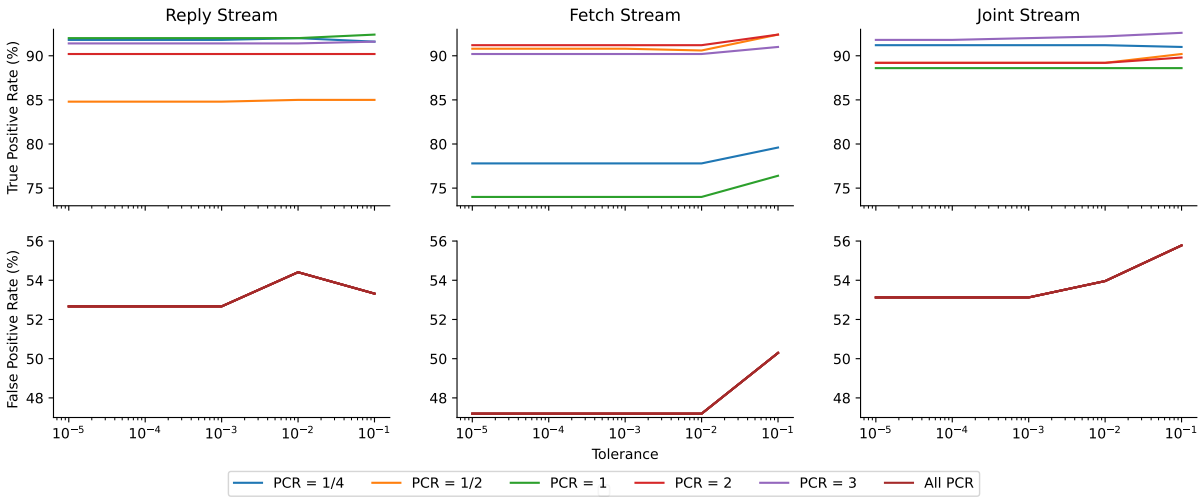

Figure 4.4: Impact of the "Tolerance" parameter on the true and false positive rates of DissecTor running with OCSVM in the otherwise benchmark configuration.

As for the different PCR values, the figure shows that, although they perform similarly for large "Nu" values, at the other end of the scale, the one we are most interested in, there seems to be an advantage in using a larger amount of probes. It is noteworthy, however, that the three smaller PCR values, the ones we would deem the most covert, achieved very similar TP rates between them, which might indicate that it could be worth using smaller values when considering $PCR \leq 1$.

Tolerance

The "Tolerance" parameter allows us to control the depth to which the algorithm runs as it controls the stopping criterion. Its value is a float whose default is $1\mathrm{e}{-3}$. In our experiments, we ran the benchmark configuration, i.e., the one containing all default parameters and no standard scaling, but varied the value of "Tolerance" logarithmically between $1\mathrm{e}{-1}$ and $1\mathrm{e}{-5}$.

Figure 4.4 presents the TP and FP rates obtained by the benchmark configuration of the OCSVM-based version of DissecTor when varying the "Tolerance". Each column of plots in the figure shows the results pertaining to a stream in the order used so far. The top row of plots shows the TP rates of each PCR as a function of the "Tolerance" and the bottom row of plots depicts the FP rate, shared by every PCR, as a function of "Tolerance". In this particular case it is noteworthy that the scale used for "Tolerance" is logarithmic. As for the color scheme it is the same as in Figure 4.3.

From this figure, we can observe that the impact of this parameter is largely insignificant and is mainly perceivable when it reaches values $> 1\mathrm{e}{-3}$. For "Tolerance" $> 1\mathrm{e}{-3}$ we can observe a slight increase in both TP and FP rates in almost every PCR-Stream pair, which might be a consequence of the model performing a smaller number of iterations to approximate the optimal solution and therefore ending up classifying a greater number of samples as anomalies.

Once more it does not seem to exist a direct relation between larger PCR values and improved detection performance. In fact, although the larger PCRs consistently placed high in terms of detection rates, there were some instances in each stream where they were outperformed by smaller ones. Given

PCR	Standard Scaling	Gamma	Tolerance	Nu	TP%	FP%	F1-score
$\frac{1}{4}$	No	scale	$\leq 1e{-3}$	0.45	90.4	46.6	0.846
$\frac{1}{2}$	No	scale	$1e{-3}$	0.15	79.8	16.2	0.849
1	No	scale	$\leq 1e{-2}$	0.35	88.2	37.1	0.853
2	No	scale	$1e{-5}$	0.05	84.6	10.6	0.891
3	No	scale	$1e{-5}$	0.05	88.2	10.6	0.912

Table 4.7: Best parameter setup for each PCR using the Reply stream according to the F1-score metric.

PCR	Standard Scaling	Gamma	Tolerance	Nu	TP%	FP%	F1-score
$\frac{1}{4}$	No	auto	$\leq 1e{-1}$	≤ 0.95	100	100	0.800
$\frac{1}{2}$	No	scale	$1e{-1}$	0.6	94.8	56.8	0.850
1	No	scale	$\leq 1e{-3}$	0.95	100	96.8	0.805
2	No	scale	$1e{-1}$	0.25	86.2	29.7	0.858
3	No	scale	$\leq 1e{-3}$	0.25	85.8	24.5	0.866

Table 4.8: Best parameter setup for each PCR using the Fetch stream according to the F1-score metric.

PCR	Standard Scaling	Gamma	Tolerance	Nu	TP%	FP%	F1-score
$\frac{1}{4}$	No	scale	$\leq 1e{-3}$	0.4	88.2	38.8	0.850
$\frac{1}{2}$	No	scale	$1e{-3}$	0.15	83.8	18.3	0.869
1	No	scale	$\leq 1e{-2}$	0.35	87.2	34.7	0.853
2	No	scale	$1e{-3}$	0.15	86.2	18.3	0.882
3	No	scale	$1e{-3}$	0.05	89.8	8.2	0.926

Table 4.9: Best parameter setup for each PCR using the Joint stream according to the F1-score metric.

the small impact increasing the "Tolerance" has on TP rates and the fact that it results in increases in already rather large FP rates, we conclude that smaller values of this parameter might be beneficial.

Best results

So far we have analyzed the impact each parameter has on system performance when isolated. We were able to reach some conclusions regarding the best values for each of them separately but are yet to show if combining them will result in better overall performance.

In this section we intend to do just that but following a different approach: instead of combining parameters and checking if the impact on the detection metrics matches our prediction we will search all experiments' results for the best ones and check if the configurations that achieved them follow the tendencies we have been identifying. Both F1-score and P4 were used to make selections and, therefore, a comparison between the two resulting sets will follow.

When several configurations obtained the same score they were analysed in order to determine their similarities. In every case where a tie between configurations occurred, with one exception, our analysis showed that it was due to the variation in "Tolerance" below a certain threshold being insignificant. Those cases are reflected in the tables.

Tables 4.7- 4.9 show the best configurations when performing evaluation with the F1-score metric. From these tables we can conclude that our predictions were, for the most part, correct as we see that: i) no configurations used pre-processing with standard scaling; ii) all configurations except one used

PCR	Standard Scaling	Gamma	Tolerance	Nu	TP%	FP%	P4
$\frac{1}{4}$	No	scale	$1e{-5}$	0.05	75.4	10.6	0.790
$\frac{1}{2}$	No	scale	$1e{-3}$	0.1	78.2	11.9	0.802
1	No	scale	$1e{-3}$	0.1	75.4	11.9	0.784
2	No	scale	$1e{-5}$	0.05	84.6	10.6	0.850
3	No	scale	$1e{-5}$	0.05	88.2	10.6	0.874

Table 4.10: Best parameter setup for each PCR using the Reply stream according to the P4 metric.

PCR	Standard Scaling	Gamma	Tolerance	Nu	TP%	FP%	P4
$\frac{1}{4}$	No	scale	$\leq 1e{-1}$	0.25	66.2	27	0.665
$\frac{1}{2}$	No	scale	$\leq 1e{-3}$	0.25	81.8	24.5	0.772
1	No	scale	$\leq 1e{-2}$	0.2	68.2	21.9	0.699
2	No	scale	$\leq 1e{-3}$	0.25	84	24.5	0.787
3	No	scale	$\leq 1e{-4}$	0.05	73.8	5.4	0.800

Table 4.11: Best parameter setup for each PCR using the Fetch stream according to the P4 metric.

PCR	Standard Scaling	Gamma	Tolerance	Nu	TP%	FP%	P4
$\frac{1}{4}$	No	scale	$1e{-3}$	0.05	74.2	8.2	0.792
$\frac{1}{2}$	No	scale	$1e{-3}$	0.15	83.8	18.3	0.812
1	No	scale	$\leq 1e{-4}$	0.1	78.6	15	0.792
2	No	scale	$\leq 1e{-4}$	0.1	84.4	15	0.830
3	No	scale	$1e{-3}$	0.05	89.8	8.2	0.895

Table 4.12: Best parameter setup for each PCR using the Joint stream according to the P4 metric.

"scale" to calculate "Gamma"; iii) tolerance values were, with some exceptions in the Fetch stream, $\leq 1e{-2}$ and; iv) again with the exception of the Fetch stream, most "Nu" values were smaller than the default (0.5), sometimes significantly.

In general the TP rates of these configurations were extremely high (from 79.8% to 100%) even though some of the FP rates were so as well (particularly in the Fetch stream), ranging all the way from 8.2% to 100%. The existence of these discrepancies, and that of large FP rates among the "best" results, not only confirms our prediction related to inadequacy of the Fetch stream for this type of analysis but is a consequence of the F1-score metric. This metric's dependency on precision and recall alone seems to make it disregard to some extent the importance of True and False negatives in its evaluation. Consequently, configurations that attained extremely large TP rates (recall) were deemed as the best despite having FP rates we find unacceptable in a system that is to be deployed in the real world.

The best results according to the P4 metric are presented in Tables 4.10-4.12. Analysing these tables we find some similarities and some differences from their F1-score counterparts. We can confirm many of the tendencies pointed out throughout this chapter, as was the case with the previous metric. However the results obtained this time around seem to support our observations more clearly: i) still none of the configurations used the pre-processing step; ii) now, all configurations used "scale" to calculate "Gamma"; iii) most "Tolerance" values were still $\leq 1e{-2}$ and; iv) the overall values of "Nu" are not only lower than the default but lower overall when compared to the F1-score results.

In general both TP and FP rates decreased when comparing to the previous metric. The TP rate now

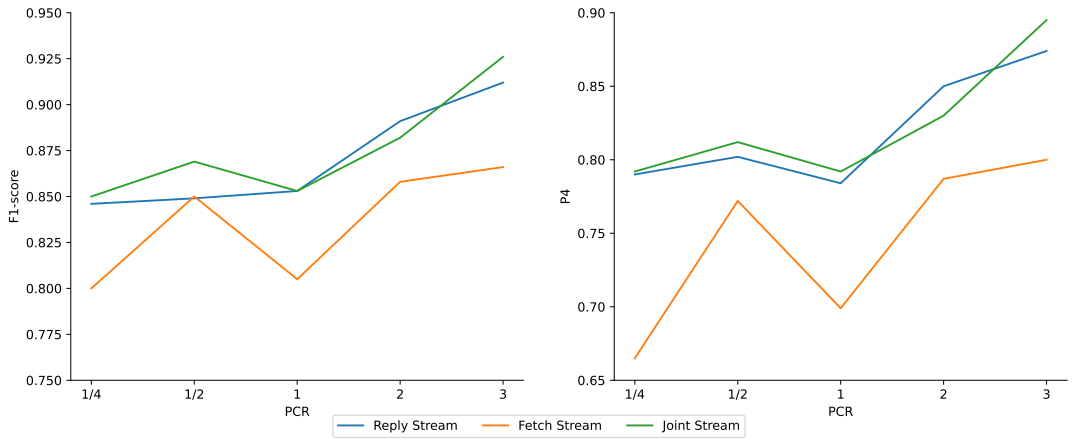

Figure 4.5: Best F1-score and P4 values of all three streams per PCR when using OCSVM.

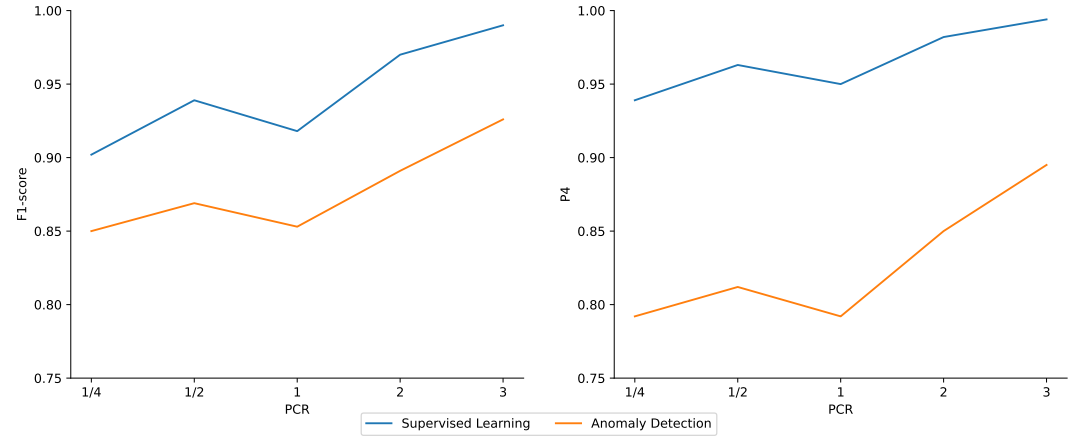

Figure 4.6: Comparison between the best F1-score and P4 values of both versions of DissecTor.

ranges from 66.2% to 89.8% while FP rates go from 5.4% to 27%. These values not only confirm that P4 results in more balanced results but also that many configurations of this version of DissecTor can be obtained following defined and simple parameter guidelines that are able to achieve good performances in regards to TP and FP simultaneously.

Figure 4.5 shows the best results achieved with each stream for every PCR when evaluated for both F1-score and P4. Besides confirming the ineptitude of the Fetch stream (in orange) as a reliable source of data to perform watermark detection it shows both the Reply and Joint streams boast very similar results. The fact that these are the best-performing streams across all versions of the system comes as no surprise as these are the ones containing more traffic data, a direct consequence of the OS's responses being larger than the requests originating them. Furthermore, the figure shows how the insight regarding the performance degradation for $PCR = 1$ is valid across all versions of the system as it is also visible in this case.

Another conclusion derived from the analysis of the best configurations and supported by Figure 4.5 resides in the fact that low PCR configurations of DissecTor may be successfully used when applying this technique as they achieved results that, not being the overall best, were not significantly worse than the ones where a larger amount of probes was used.

Finally, in Figure 4.6, we present a comparison of the best results of each watermark detection technique evaluated in this chapter for each PCR value. The figure shows that, although the best configurations of the OCSVM version of the system (in orange) still lack in comparison to their supervised learning counterparts (in blue), we can still consider it a viable alternative for scenarios where the latter is impractical. Actually, it is important to note that it is expected that the vast majority of the attacks resort to the anomaly detection technique, as the fact that the supervised learning approach relies on mock scenarios using extremely difficult to attain information makes them the exception and not the rule.

4.3.3 Summary

This chapter presented results of several experiments conducted on DissecTor with the intention of evaluating its performance. Both methodologies of watermark detection were evaluated and several variants of each of them were analysed. The results were interpreted having DissecTor's two main goals in mind: i) successful and reliable detection of the watermark and; ii) covertness.

In particular we showed how the supervised learning version of the system achieved very good TP and FP rates even for low values of PCR, despite the results clearly showing that a larger amount of probes results in better performance.

While analysing the OCSVM version of DissecTor we showed how we are able to empirically derive the best combination of parameters by individually studying the impact each one has. In the end we presented two sets of "best" configurations according to two different metrics and both seem to confirm our previous observations. From the two "best" sets we would like to highlight the one obtained when using the P4 metric since it seems to strike a good balance between TP and FP rates. The results shown in these configurations allow us to conclude that, although larger PCR values tend to perform consistently better, the relation between PCR and the performance is not as significant as in the supervised learning version of the system. In fact we concluded that lower PCR, more covert, configurations present satisfactory results and might be viable.

We concluded by comparing the two watermark detection methods, noting that supervised learning was the best alternative when the ideal circumstances presented themselves although anomaly detection constituted a viable method for when they did not. It is noteworthy, however, the fact that, because these ideal conditions are extremely rare – one can not expect to have privileged information regarding the target's number of clients, their access patterns and so on – the second version of the system, based on anomaly detection, is much more realistic.

Chapter 5

Conclusions

Tor and its Onion Service infrastructure allow users to interact with Internet services while benefiting from sender and receiver anonymity. As a consequence, Tor has gained popularity both among those seeking protection from authoritarian/suppressive regimes and those looking to conduct illegal activities without being identified. Law-enforcement's interest in breaking Tor's anonymity guarantees has been growing, particularly in what deals with the identification of Onion Services providing unlawful content.

This thesis presented DissecTor, a distributed system whose goal is to allow LEAs to identify specific OSes' IP addresses, towards aiding digital investigation efforts. DissecTor assumes a model where multiple LEAs around the world cooperate and form coalitions so as to create a global network adversary. Taking advantage of this global adversary model, we proposed DissecTor, an active traffic correlation attack based on an acceleration-based watermarking technique. DissecTor makes use of different machine learning techniques to perform watermark detection and focuses on both accuracy and covertness.

5.1 Achievements

This work produced the following outcomes:

- We presented DissecTor, a system capable of targeting and deanonymizing specific Tor OSes. DissecTor leverages active traffic correlation attacks based on flow watermarking with machine learning techniques to perform watermark detection.

- We proposed and presented a new acceleration-based watermarking scheme to conduct attacks on Tor OSes. This technique is innovative in the sense that it combines approaches of both frequency and space-based watermark diversity schemes while using a rate-based carrier and non-blind detection method.

- We designed and explored two different versions of DissecTor, each employing a different machine learning technique to perform watermark detection.

- We performed a thorough evaluation of DissecTor's two watermark detection methodologies and determined the use cases where each one is more effective than the other. Furthermore, we explored several variants for each of the two methodologies and determined the best configurations for each one while showing the system is able to achieve satisfactory detection results and covertness/stealthiness properties.

5.2 Future Work

This section presents multiple avenues for future work on improving our DissecTor active watermarking scheme:

Experimenting with different numbers of probe requests: The first aspect concerns the amount of requests sent by each probe when conducting a probing session. In our experiments we had each probe send 5 consecutive requests to the target. This was done as a watermark amplification measure and a 1 request version of the system was not evaluated.

Improved watermarking via OS multi-page browsing: The second aspect is related to the first and consists of an idea for improving the system in both effectiveness and covertness. It would be interesting to explore if DissecTor could successfully be adapted in order to request specific OS web pages other than the landing one, used throughout this document. By studying each target more carefully beforehand and giving DissecTor a set of pages to access in succession we might not only be able to justify the use of a larger amount of requests per probing session (further amplifying the watermark) but also make the system behave closer to a regular client while tailoring the watermark further to each OS.

Realistic testbed with multiple OSes: The third aspect concerns another testing scenario where DissecTor is yet to be evaluated and consists in performing attacks in more realistic circumstances. The experiments conducted until this point focus on the system's ability to identify watermarked traffic when a single OS's traffic is tested. A more realistic scenario should be explored in which several OSes' traffic is analysed and a single target must be identified from the set of candidates.

Extensive covertness evaluation: Finally we would like to devise experiments allowing for a more objective evaluation of DissecTor's covertness as the one done so far is mainly based on reducing the attack's "footprint" – mainly reducing the amount of probes used in the watermarking stage.

Bibliography

[1] The Tor Project. History. https://www.torproject.org/about/history/. Accessed: 2022-31-10.

[2] Freedom of the Press Foundation. SecureDrop. https://securedrop.org/. Accessed: 2022-01-10.

[3] A. Greenberg. ProPublica Launches the Dark Web's First Major News Site. https://www.wired.com/2016/01/propublica-launches-the-dark-webs-first-major-news-site/, July 2016. Accessed: 2022-31-10.

[4] BBC News. BBC News launches 'dark web' Tor mirror. https://www.bbc.com/news/technology-50150981. Accessed: 2022-31-10.

[5] M. Reed, P. Syverson, and D. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected areas in Communications*, 16(4):482–494, 1998.

[6] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr. Towards an analysis of onion routing security. In *Designing Privacy Enhancing Technologies*, pages 96–114. Springer, 2001.

[7] P. Winter and S. Lindskog. How china is blocking tor. *arXiv preprint arXiv:1204.0447*, 2012.

[8] N. Christin. Traveling the silk road: a measurement analysis of a large anonymous online marketplace. In *Proceedings of the International Conference on World Wide Web*, 2013.

[9] K. Soska and N. Christin. Measuring the longitudinal evolution of the online anonymous marketplace ecosystem. In *Proceedings of the USENIX Security Symposium*, 2015.

[10] R. Hurley, S. Prusty, H. Soroush, R. J. Walls, J. Albrecht, E. Cecchet, B. N. Levine, M. Liberatore, B. Lynn, and J. Wolak. Measurement and analysis of child pornography trafficking on p2p networks. In *Proceedings of the International Conference on World Wide Web*, 2013.

[11] G. Weimann. Going dark: Terrorism on the dark web. *Studies in Conflict & Terrorism*, 39(3): 195–206, 2016.

[12] M. Casenove and A. Miraglia. Botnet over tor: The illusion of hiding. In *Proceedings of the International Conference On Cyber Conflict*, 2014.

[13] B. Evers, J. Hols, E. Kula, J. Schouten, M. den Toom, R. van der Laan, and J. Pouwelse. Thirteen years of tor attacks. URL `https://github.com/Attacks-on-Tor/Attacks-on-Tor`. https://github.com/Attacks-on-Tor/Attacks-on-Tor.

[14] M. Nasr, A. Bahramali, and A. Houmansadr. Deepcorr: Strong flow correlation attacks on tor using deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1962–1976, 2018.

[15] F. Rezaei and A. Houmansadr. Finn: Fingerprinting network flows using neural networks. In *Proceedings of the Annual Computer Security Applications Conference*, pages 1011–1024, 2021.

[16] P. Medeiros. Distributed system for cooperative deanonymization of tor circuits, January 2021.

[17] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[18] N. S. Evans, R. Dingledine, and C. Grothoff. A practical congestion attack on tor using long paths. In *Proceedings of the USENIX Security Symposium*, pages 33–50, 2009.

[19] S. Chakravarty, A. Stavrou, and A. D. Keromytis. Traffic analysis against low-latency anonymity networks using available bandwidth estimation. In *European symposium on research in computer security*, pages 249–267. Springer, 2010.

[20] X. Wang, J. Luo, M. Yang, and Z. Ling. A potential http-based application-level attack against tor. *Future Generation Computer Systems*, 27(1):67–77, 2011.

[21] S. J. Murdoch and P. Zieliński. Sampled traffic analysis by internet-exchange-level adversaries. In *Proceedings of the International workshop on privacy enhancing technologies*, pages 167–183. Springer, 2007.

[22] M. Edman and P. Syverson. As-awareness in tor path selection. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 380–389, 2009.

[23] R. Nithyanand, O. Starov, A. Zair, P. Gill, and M. Schapira. Measuring and mitigating as-level adversaries against tor. *arXiv preprint arXiv:1505.05173*, 2015.

[24] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker. Low-resource routing attacks against tor. In *Proceedings of the 2007 ACM workshop on Privacy in electronic society*, pages 11–20, 2007.

[25] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal. Raptor: Routing attacks on privacy in tor. In *Proceedings of the 24th USENIX Security Symposium (USENIX Security 15)*, pages 271–286, 2015.

[26] B. N. Levine, M. K. Reiter, C. Wang, and M. Wright. Timing attacks in low-latency mix systems. In *Proceedings of the International Conference on Financial Cryptography*, pages 251–265. Springer, 2004.

[27] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy (S&P'05)*, pages 183–195. IEEE, 2005.

[28] Y. Zhu, X. Fu, B. Graham, R. Bettati, and W. Zhao. Correlation-based traffic analysis attacks on anonymity networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(7):954–967, 2009.

[29] S. Chakravarty, A. Stavrou, and A. D. Keromytis. Linkwidth: a method to measure link capacity and available bandwidth using single-end probes. 2008.

[30] X. Wang and D. S. Reeves. Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 20–29, 2003.

[31] A. Houmansadr, N. Kiyavash, and N. Borisov. Rainbow: A robust and invisible non-blind watermark for network flows. In *Proceedings of the Network and Distributed Systems Security Symposium*, 2009.

[32] A. Iacovazzi, S. Sarda, and Y. Elovici. Inflow: Inverse network flow watermarking for detecting hidden servers. In *Proceedings of the IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 747–755, 2018. doi: 10.1109/INFOCOM.2018.8486375.

[33] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas. Circuit fingerprinting attacks: Passive deanonymization of tor hidden services. In *Proceedings of the 24th USENIX Security Symposium (USENIX Security 15)*, pages 287–302, 2015.

[34] A. Panchenko, A. Mitseva, M. Henze, F. Lanze, K. Wehrle, and T. Engel. Analysis of fingerprinting techniques for tor hidden services. In *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*, pages 165–175, 2017.

[35] X. Wang, D. S. Reeves, S. F. Wu, and J. Yuill. Sleepy watermark tracing: An active network-based intrusion response framework. In *Proceedings of the IFIP International Information Security Conference*, pages 369–384. Springer, 2001.

[36] A. Iacovazzi and Y. Elovici. Network flow watermarking: A survey. *IEEE Communications Surveys & Tutorials*, 19(1):512–530, 2016.

[37] A. Houmansadr and N. Borisov. Botmosaic: Collaborative network watermark for the detection of irc-based botnets. *Journal of Systems and Software*, 86(3):707–715, 2013.

[38] A. Houmansadr and N. Borisov. Swirl: A scalable watermark to detect correlated network flows. In *Proceedings of the NDSS Symposium*, 2011.

[39] W. Yu, X. Fu, S. Graham, D. Xuan, and W. Zhao. Dsss-based flow marking technique for invisible traceback. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy (SP'07)*, pages 18–32. IEEE, 2007.

[40] J. Huang, X. Pan, X. Fu, and J. Wang. Long pn code based dsss watermarking. In *Proceedings of the 2011 Proceedings IEEE INFOCOM*, pages 2426–2434. IEEE, 2011.

[41] Z. Ling, X. Fu, W. Jia, W. Yu, D. Xuan, and J. Luo. Novel packet size-based covert channel attacks against anonymizer. *IEEE Transactions on Computers*, 62(12):2411–2426, 2012.

[42] X. Luo, P. Zhou, J. Zhang, R. Perdisci, W. Lee, and R. K. Chang. Exposing invisible timing-based traffic watermarks with backlit. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 197–206, 2011.

[43] Z. Lin and N. Hopper. New attacks on timing-based network flow watermarks. In *Proceedings of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 381–396, 2012.

[44] A. Iacovazzi, D. Frassinelli, and Y. Elovici. The duster attack: Tor onion service attribution based on flow watermarking with track hiding. In *Proceedings of the 22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*, pages 213–225, 2019.

[45] J. Hayes and G. Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *Proceedings of the 25th USENIX Security Symposium (USENIX Security 16)*, pages 1187–1203, 2016.

[46] D. Barradas, N. Santos, and L. Rodrigues. Effective detection of multimedia protocol tunneling using machine learning. In *Proceedings of the 27th USENIX Security Symposium (USENIX Security 18)*, pages 169–185. USENIX Association, Aug. 2018.

[47] T. T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE communications surveys & tutorials*, 10(4):56–76, 2008.

[48] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg. Effective attacks and provable defenses for website fingerprinting. In *Proceedings of the 23rd USENIX Security Symposium (USENIX Security 14)*, pages 143–157, 2014.

[49] R. Zhang, S. Zhang, S. Muthuraman, and J. Jiang. One class support vector machine for anomaly detection in the communication network performance data. In *Proceedings of the 5th conference on Applied electromagnetics, wireless and optical communications*, pages 31–37. Citeseer, 2007.

[50] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[51] F. Eibe, M. A. Hall, and I. H. Witten. The weka workbench. online appendix for data mining: practical machine learning tools and techniques. In *Morgan Kaufmann*. Morgan Kaufmann Publishers, 2016.

[52] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[53] Y. Sasaki et al. The truth of the f-measure. *Teach tutor mater*, 1(5):1–5, 2007.

[54] M. Sitarz. Extending f1 metric, probabilistic approach. *arXiv preprint arXiv:2210.11997*, 2022.

[55] R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[56] Scikit-learn. RBF SVM parameters. `https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html`, . Accessed: 2022-26-10.

[57] Scikit-learn. sklearn.svm.OneClassSVM. `https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html`, . Accessed: 2022-26-10.

[58] M. Zhang, B. Xu, and D. Wang. An anomaly detection model for network intrusions using one-class svm and scaling strategy. In *Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 267–278. Springer, 2015.

[59] G. H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, pages 338–345. Morgan Kaufmann, 1995.

[60] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.