Towards De-anonymizing Hidden Services

Master Thesis

MSc Cyber Security, 9th & 10th sem.

Clifford-Nelly Ndayikengurukiye

Aalborg University
Department of Electronic Systems
Fredrik Bajers Vej 7B
DK-9220 Aalborg Ø

Department of Electronic Systems
Fredrik Bajers Vej 7B
DK-9220 Aalborg Ø
https://es.aau.dk

AALBORG UNIVERSITY

Title:
De-anonymizing Hidden Services

Theme:
Long Master Thesis

Project Period:
Spring semester 2022 & Fall semester 2023

Project Group:

Participants:
Clifford-Nelly Ndayikengurukiye

Supervisors:
Jens Myrup Pedersen
Shreyas Srinivasa

Copies:

Number of Pages: 79

Date of Completion:
June 2, 2023

Abstract:

This thesis explores the difficulties and limitations of attempting to de-anonymize hidden services. The method of attack proposed relies on monitoring watermarked packets from the target's compromised guard.
Hidden services are a key feature of Tor that offer anonymity by concealing the IP addresses of services. The primary obstacle arises from the inherent design principles of the Tor network, which ensures user privacy through multi-layered encryption.
Tor cells, the fundamental units of data transmission within the network, are uniformly encrypted and share identical sizes, making it difficult to differentiate between them based on packet characteristics alone.
This research emphasizes the robustness of Tor's privacy mechanisms and highlights the importance of addressing potential threats to user anonymity.

Nomenclature

Abbreviation	*Name*
Tor	The Onion Router
THS	Tor Hidden Service
MitM	Man-in-the-Middle
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
DoS	Deniel of Service
HSDir	Hidden Service Directory
E2EE	End-to-end Encryption
E2E	End-to-end
IP	Internet Protocol
TCP	Transmission Control Protocol
DHT	Distributed Hash-Table
GUI	Graphical User Interface
API	Application Programming Interface
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
TLS	Transport Layer Security
PoC	Proof-of-Concept
JS	JavaScript
JSON	JavaScript Object Notation
IEEE	Institute of Electrical and Electronics Engineers
CLI	Command-line Interface
C2	Command-and-Control
ISP	Internet Service Provider
AI	Artificial Intelligence
SOTA	State-of-The-Art
RP	Rendezvous Point
ORPort	Onion Router Port
LAN	Local Area Network
UFW	Uncomplicated Firewall
IaaS	Infrastructure-as-a-Service
VM	Virtual Machine
AES	Advanced Encryption Standard
DPI	Deep Packet Inspection

Contents

Preface

Clifford-Nelly Ndayikengurukiye
<cndayi21@student.aau.dk>

Aalborg University, June 2, 2023

Introduction 1

Developed in the mid-1990s, Tor is an open-source software technology primarily used to anonymize online communications[1]. The ubiquity of the Internet necessitates privacy. Due to this, anonymity becomes paramount in online technological advancements. The need for anonymity permeates online communications. The most prominent network of the aforementioned is Tor.

The Tor network is geared towards anonymity allowing users to post and browse content online anonymously. An anonymous network such as Tor enables users like human rights advocates and whistle-blowers to promote and securely engage in anti-censorship causes. Although the anonymity provided by Tor is positive, excessive illegal active becomes prominent as far as Tor Hidden Services (THS) are concerned[2]. THSs are a key feature of the dark web. When a web application is added to the Tor network it becomes a THS. In layman's terms, a THS is an anonymized application server.

As of 2015, Figure 1.1 shows that every day about 30000 THSs announced themselves to the THS directories[3].

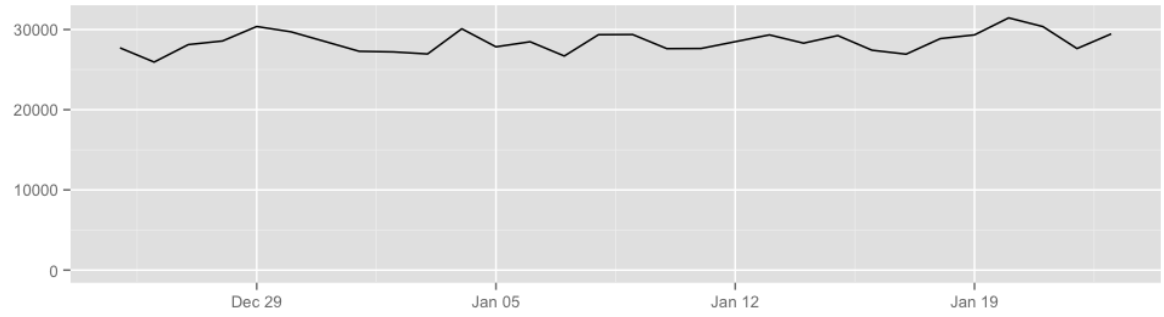

Figure 1.1: Number of THSs in 2015 [3].

THSs are considered to be a controversial aspect of the dark web and is the topic of much debate[2]. The prominent cybersecurity company, Kaspersky, defines the dark web as follows:

> *"The dark web is the hidden collective of internet sites only accessible by a specialized web browser. It is used for keeping internet activity anonymous and private, which can be helpful in both legal and illegal applications. While some use it to evade government censorship, it has also been known to be utilized for highly illegal activity[4]."*

The anonymity the Tor network provides can facilitate the purchase of illegal items, substances and services[2]. Furthermore, THSs are primarily meant to provide anonymity to darknet

administrators and vendors. As entire transactions have End-to-end encryption (E2EE), Tor has become a safe haven for nefarious activities. Therein lies the dilemma, however. By de-anonymizing THSs, one may inadvertently put journalists and activists in politically hostile territories at risk of discovery and thus undermine Tors initial purpose. The purpose in question is articulated in the following statement.

> *"Just like Tor users, the developers, researchers, and founders who've made Tor possible are a diverse group of people. But all of the people who have been involved in Tor are united by a common belief:* ***internet users should have private access to an uncensored web[1].*** "

Therefore, it must be stated that the approaches of de-anonymization presented in this paper are purely meant to be viewed for research purposes.

In 2015, a study showed the amount of Tor traffic THSs were responsible for. The preliminary results in Figure 1.2 showed that THSs cause somewhere between 400 to 600 Mbit of traffic per second, or equivalently about 4.9 terabytes a day[3].

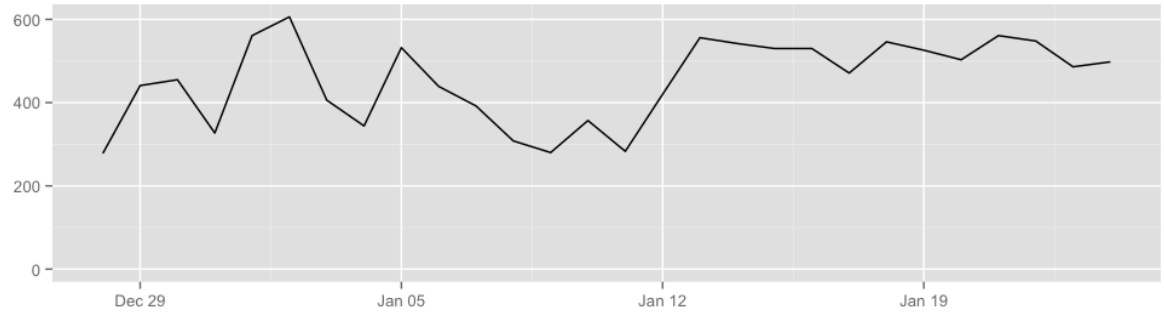

Figure 1.2: THS traffic in 2015 [3].

These statistics show the excessive usage of THSs and how they can be problematic in nature. Law enforcement and hackers therefore continuously target these THSs. This thesis' aim is to analyze current de-anonymization methods and assess their effectiveness through trial and error. In doing this, a comprehensive approach can be derived and potentially become reproducible for future research.

1.1 Problem Statement

The aim of this section is to clearly define a set of questions meant to be answered throughout this paper. This will ultimately define the scope and parameters the paper must adhere to. First, a broad, overarching and all-encompassing question must be defined.

- Which methodologies and techniques exist when de-anonymizing THSs, and how can such methods be improved?

Next, a collection of smaller sub-questions will be defined to narrow down the scope of the paper further.

- How does a THS operate on the Tor network?
- What can be gathered from previous research on de-anonymizing THSs?
- How do existing methods compare in terms of applicability and robustness?
- How can de-anonymization techniques be improved?

Methodology 2

This chapter describes the various processes and approaches used throughout this paper. The search and evaluation of relevant literature will be described. This includes the presentation of identified themes and literary gaps.

The conclusions drawn from the literary and practical approach will be further emphasized in the literature review in Chapter 3.

2.1 Approach

This section will showcase the processes and approach in which, the de-anonymization of a THS will take place. Additionally, the methodologies in which the literature review was conducted will be described.

The literary approach to this thesis can be described as a comparative case study of the various existing methods of de-anonymization. Although an analytical focus will be placed on each literary source, they will merely serve as a basis for the upcoming practical portion of the paper.

2.1.1 Literary

In order to acquire and gauge the maturity of current academic literature, a literature review is conducted. By conducting a literature review, an analytical process can commence in which a range of academic sources are assessed so that themes and literary gaps can be derived. In addition, a discussion would determine their significance to the problem scope.

By critically evaluating and analyzing the source material, a theoretical basis can be established for potential attack vectors as seen in Section 3.1. Additionally, identifying current literary gaps on the subject and discussing their significance ensures that the project contributes with new knowledge to the field.

Search Process

Initially, a method of searching for academic sources had to be derived. This would follow an evaluation of their quality and significance to the subject. Searches were done using prominent means such as academic publishing search engines akin to Google Scholar [5] & IEEE. Applicable search terms were used such as *"De-anonymizing hidden services", "De-anonymizing Tor", "De-anonymizing Onion Services", "Attacking the Darkweb", "Attacking the Darknet", "Attacking Tor", "Attacking Onion Services", "Hacking the Darkweb", "Hacking the Darknet", "Hacking Tor", "Hacking Onion Services", "Tor", "Onion Routing", "Darkweb", "Darknet", "Anonymity", "De-anonymizing the darknet" & "De-anonymizing the darkweb".*

During this process **20 relevant published papers were identified** on the basis of the following:

- **Citation quantity:** Papers were considered as long as the number of citations were reasonable (over 10).
- **Title relevance:** Papers were considered as long as the titles were relevant to the keywords searched.
- **Publishing dates:** Papers were considered as long as the publication date was recent (after 2017).

The publishing date criteria was considered to ensure that older papers were filtered out. This would ensure the technologies involved were not outdated. Similarly, the number of citations criteria was used to gauge the validity of the papers e.g., a lesser paper would likely not be continuously referenced by the scientific community. At that point, each of the 20 papers were subjected to screening and evaluation.

Evaluation Process

This process consisted of evaluating an attack's feasibility, discussing its relevance to the project's scope and an assessment of the paper's quality. The relevance was determined by reading the abstract and introduction of each paper, whereas, the quality was determined by reading their methodology and conclusion.

This process is subjective and it should be noted that the evaluation only reflects the opinions of the author. In addition, these opinions are not an objective classification of research quality. Rather, this process should be identified as a specialized method to match a papers specific topic to this project. Additionally, this approach is largely dependent on evaluating the quality and significance of research papers in relation to the given context.

In most cases, establishing a citation criteria proved difficult, as papers often had few citations despite having significant merits expressed in their results. On occasion, these papers were also assessed. A combination of these evaluation parameters formed the first assessment. Following the first assessment, a new evaluation system was used to deem each paper *"accepted"* or *"rejected"*, and if accepted, the paper's attack methods would be fully examined.

Out of the 25 papers that were initially examined, 20 papers were accepted and 5 were deemed either irrelevant to the project or of too low quality. This gave an **initial acceptance rating of 80%**. Each of the 20 accepted papers were then subjected to a complete review of the content. At this point, a rating system for each attack method (presented in each paper) was evaluated based on *cost* and *impact*. *Cost*, being in reference to the pricing and feasibility of each attack method and *impact*, being in reference to the resulting damage (the level of de-anonymization e.g., the amount of information extracted). An evaluation table in Section 3.1 further emphasizes this selection process.

Finally, evaluating each paper in this way provided an overview of relevant findings and established a general knowledge base. A discussion of which literary gaps had been identified through analysis is shown in Section 3.1.

Summary

In summary, the literature search identified 25 relevant papers to the context of de-anonymizing THS. 20 of these papers were accepted after the initial screening process and further analyzed in detail. Attack methods described in the literature were assessed based on their resource dependency (*cost*) and the severity of consequences that would arise from an attack (*impact*). In addition, each attack was given a final rating ranging from low to high as summarized in Chapter 3.

2.1.2 PoC

Figure 2.1 illustrates the methodology of which was followed throughout this project from a practical perspective. The rest of this section describes each of the five phases in detail.

Figure 2.1: Overview of project structure

By using the problem statement as a pillar, the general direction of the conceptualization and scoping phases can be established.

The **conceptualization phase** is used to establish an initial idea to be manifested and serves as a basis for the envisioned approach. As this project takes a red team perspective, the development of a system to attack will ultimately be secondary. Through a high-level overview, the attack methods proposed needed to solve the problem statement devised in Section 1.1.

The aim of the **scoping phase** is to create requirements that will determine the scope of the project. These requirements pertain to both the system development and attacks. Furthermore, the scope maps out the main focus areas of the project in addition to its limitations.

The limitations are described in the **requirements phase**. This phase will establish coherent rules and objectives to be pursued throughout the process.

Having established concrete requirements for the system and attack scenarios, the **design & implementation phase** is commenced. This phase ensures conceptual models will be constructed respecting technological constraints and other measures in Section 4.3 and 5.1 respectively. Once designed, implementations of these models will be executed. Furthermore, a step by step reenactment of the difficulties involved will be expressed.

Finally, an overall **evaluation** of the results will determine the projects outcome. Additionally, a reflective view of the aforementioned difficulties will be taken followed by their evaluation.

Literature Review 3

The following section will describe detailed literature reviews of relevant source material on the topic of de-anonymizing THSs.

A literature review serves the purpose of providing a reader the with an understanding of the State-of-The-Art (SOTA) in the area. Ultimately, it serves as inspiration and helps devise parameters the project can adhere to. In essence, the literature review enables the project to continue, deviate or further the technological advancements of previous researchers. The reviews in this section will focus on the attack methods presented in each paper. However, a method to decide the value of related works must be determined.

The main factors to consider in such an undertaking would be resource availability and the impact of the attack e.g., its effectiveness and the implications involved. Therefore, it has been determined that conducting a cost benefit analysis would be the best course of action when determining a papers value to the project.

Cost benefit should be interpreted in a way where any resource, be it time, money, technology or man power is seen as a commodity that furthers the possibility of completing the projects central goal. By reviewing the literature and assessing their methods and approaches, a cost benefit comparison will clarify the parameters of what can be done. Given the resources available for the project, a comparison to the literature being reviewed will provide insight into how to best utilize resources. Each paper will be assessed based on the following criteria.

- Impact
- Cost

In this case, impact refers to the overall impact e.g., the quality and quantity of the extracted information during the de-anonymizing process. The cost, however, will not be a precise budgetary estimate but rather an evaluation of the number of required resources. The upcoming attack methods will be critiqued and receive a rating within the two criteria. These ratings will be one of the following:

- High
- Medium
- Low

These scores can be further described by the following table:

Ratings		
Score	**Impact**	**Cost**
High	Fully compromised system (root privilege)	Unavailable tools
Medium	Semi compromised system (IP leakage)	Payment for tools
Low	No information disclosure	Free tools

Ideally, an attack method would have a rating of *"High Impact"* and *"Low Cost"*.

The following section describes various attack methods in addition to their ratings, reviews and papers (papers in which the attack methods were presented).

Fingerprinting Attack

> **Paper (1):** Circuit Fingerprinting Attacks: Passive De-anonymization of Tor Hidden Services [6]
> **Impact:** Low
> **Cost:** Medium

The paper in question proposes two attacks, under two different threat models. Both attacks are fingerprinting attacks that take advantage of compromised guards. The threat models deviate when considering circuits that are established by either the client or the THS.

Ultimately, fingerprinting attack methods are passive. User involvement with hidden services had been identified with a 98% true positive rate and less than 0.1% false positive rate with the first attack, and 99% true positive rate and 0.07% false positive rate with the second. Although the success rate when de-anonymizing clients seem promising, the focus does not appear to be on de-anonymizing THSs.

By reconstructing Tor circuits, the attack methods allow for in-depth analysis of darknet activity. Although, attempts are made to de-anonymize THSs, ultimately, no IP addresses where discovered.

> **Paper (2):** Analysis of Fingerprinting Techniques for Tor Hidden Services [7]
> **Impact:** Medium
> **Cost:** Low

This paper focuses on fingerprinting Tor traffic to de-anonymize THSs. It depicts a two-phased approach for fingerprinting THSs that does not rely on malicious Tor relays. Instead, datasets are utilized to determine the likelihood of a particular THS having been visited. Phase 1 concerns *"Detection of Communication to Hidden Services"* while phase 2 cencerns *"Detection of Particular Hidden Service Content"*.

The paper preferences that these existing approaches do not scale when applied in realistic settings. Furthermore, the feasibility of the approach will likely decrease in the future as a result of the size of the hidden service universe.

Paper (3): POSTER: Fingerprinting Tor Hidden Services [8]
Impact: Low
Cost: Low

In this paper a two-phased fingerprinting approach is proposed. The first step being to recognize Hidden Service clients and the second to estimate scalability by extending the hidden service dataset.

By analyzing patterns such as packet sizes, their order, and direction this fingerprinting attack is scalable to the Tor network. However, it does not rely on malicious Tor relays. This approach is niche as it is only applicable for certain versions of Tor and less than 1.5% of connections to THSs. Ultimately, the results were limited and did not go into much detail as to how this attack was conducted.

Paper (4): Fingerprinting Hidden Service Circuits from a Tor Middle Relay [9]
Impact: Medium
Cost: Low

In this paper, fingerprinting attacks are being conducted on THSs from middle relays. Additionally, a random forest classifier is utilized for an increased accuracy of 99.98%. This is a unique approach that would garner further investigation, however, with limited results, the paper does not go into much detail as to how this attack is conducted.

Napping Guard Attack

Paper: Napping Guard: De-anonymizing Tor Hidden Service in a Stealthy Way [10]
Impact: High
Cost: Low

This paper utilized a Napping Guard attack, in which, a covert channel is built. It can send messages from a malicious guard to the client. The covert channel, allows the guard relay to deliver the actual IP address of the THS to the client. By doing so it correlates the IP address to the onion address. Although, its practical application was depicted as particularly circumstantial, the covert channel was reliable with a precision and recall rate of 99.35% and 99.19% respectively.

SignalCookie Attack

Paper: SignalCookie: Discovering Guard Relays of Hidden Services in Parallel [11]
Impact: High
Cost: Low

This paper proposes a new approach called the SignalCookie attack. The attack results in the reveal of multiple hidden service entry relays in parallel. It utilizes Rendezvous Cookies and circuit watermarks to deliver the THS's identifiers to controlled relays. By exploiting a design flaw that allows the Rendezvous Point to identify which hidden service creates circuit to it, the Rend-Point can embed the identifier into the circuit as circuit watermark. By detecting the circuit watermark, the controlled relay can recognize the identity of the hidden service.

As a newer method developed in 2019, they discovered 20% of entry relays serve 89.32% hidden services. With a significantly lower level approach, it appears to be promising. It should be noted, their experiment lasted between 7 and 17 months.

Descriptor Collecting

Paper: Trawling for Tor Hidden Services: Detection, Measurement, De-anonymization [12]
Impact: Low
Cost: High

This paper attempts to de-anonymize THSs by collecting their descriptors. There is an attempt to amplify these effects by utilizing botnets and C2 servers to flood requests. Ultimately, this approach is resource demanding and of medium in benefits. They spent 100 USD to perform this experiment. Furthermore, the experiment required two days worth of nonstop collecting of data. These costs are not unreasonable, however, they are not ideal. Additionally, the added task of researching botnet technology could be time consuming and may result in deviating from the projects initial scope.

Collection of Attacks

Paper (1): De-anonymizing schemes of hidden services in Tor network: A survey [13]
Impact: Low
Cost: Medium

Paper (2): Tor Hidden Services: A Systematic Literature Review [14]
Impact: Low
Cost: Low

These papers are accumulations of various attack methods on THSs. Overall, these methods are low in cost, however, not necessarily reproducible. Although relevant, these methods do not go in depth enough nor demonstrate any practical examples of de-anonymizing THSs. The results are limited as the papers only serve as a collection of ideas. It is apparent the papers only purpose is to serve as inspiration for researches as opposed to an in-depth guide. Many of the ideas expressed in these papers are elaborated more extensively in other papers listed in this literature review.

CARONTE Tool

Paper: CARONTE: Detecting Location Leaks for De-anonymizing Tor Hidden Services [15]
Impact: Low
Cost: Low

This paper utilizes the CARONTE tool to identify location leaks in THSs. It does not require a data-set of potential servers in advance. After visiting the THS, CARONTE extracts Internet endpoints and looks up unique strings from the THS's content. In addition, it examines the THS's certificate chain to extract potential Internet endpoints e.g., server locations. Although an interesting tool, CARONTE does not seem to be available at the time of writing this paper.

Sniper Attack

Paper: The Sniper Attack: Anonymously De-anonymizing and Disabling the Tor Network [16]
Impact: High
Cost: Medium

This paper utilizes a Sniper Attack. A Sniper Attack is a low cost but destructive DoS attack. Its level destruction is measured by its ability to reducing Tor's bandwidth capacity by 35%.

It enables attackers to anonymously disable arbitrary Tor relays. The attack utilizes valid protocol messaging to consume memory. This is done by exploiting Tor's end-to-end reliable data transport. Although an interesting project, this paper focuses to heavily on disabling THS traffic as opposed to de-anonymization.

MitM Attack

Paper: Off-path man-in-the-middle attack on Tor hidden services [17]
Impact: Medium
Cost: Low

This paper performed a MitM attack by compromising the THS's private key. The adversary does not need to be in the communication path between the client and the service. The results are limited as the paper does not go into detail of how it compromises the private key.

Although the paper focused on client deception, it did not fully neglect the de-anonymization of THSs. The paper is not overly detailed as to how this attack is conducted. Finally, although not outright stated, the paper did showcase potential avenues for de-anonymizing THSs.

Inflow Attack

Paper: Inflow: Inverse Network Flow Watermarking for Detecting Hidden Servers [18]
Impact: High
Cost: Low

This paper utilizes an inflow attack method. Inflow is a new technique to identify THSs based on inverse flow watermarking. It is a new watermarking technique used to link illegal content made available on Tor to the THS providing the content.

Inflow drops bursts of packets for short time intervals, ultimately resulting in time gaps in flows observed on the receiving side of a traffic flow coming from the THS. The packets in question are ACK packets. They are dropped in the opposite direction of the data traffic (from client to guard) at pseudo random times. This occurs so a sequence of silent time-intervals can compose a watermark and embed it. The watermark can be recognized by a detector that knows a secret key and is placed along the path from THS to the THS's guard. Finally, by controlling the communication edges and detecting the watermarking gaps, Inflow can detect THSs.

The results obtained show true positive rates in the range of 90 to 98%. An effective and low level approach, it additionally requires minimal resources.

Exploit Tor v2 Flaw

Paper: A Tool to Extract Onion Links from Tor Hidden Services and Identify Illegal Activities [19]
Impact: Low
Cost: Low

This paper presents an attack method that exploits a flaw in the Tor v2 protocol to collect onion links of THSs from the memory of a THS Directory. This paper is not particularly relevant as it does not demonstrate the deanonymization of THSs but rather, whether or not illegal activities are taking place and the type.

Bitcoin Transfer Analysis

Paper: Darknet and Bitcoin De-anonymization: Emerging Development [20]
Impact: High
Cost: High

This paper attempts to de-anonymize clients and THSs by analyzing bitcoin transactions. This approach is practical, however, it is resource consuming and lands outside the scope of this project. Additionally, bitcoin technology covers enough material to carry an entirely separate project.

This would be a more complex avenue to take and would require an entire reshaping of the exesting project. In addition to being time consuming, this would require researching and

studying entirely new categories and topics in academic literature. Although, new research material is advantageous, it would require expanding the initial scope to an excessive degree.

Journalism (No Method)

Paper: Investigative Techniques for the De-Anonymization of Hidden Services [21]
Impact: Low
Cost: Low

The paper is not particularly technical and approaches the topic from a journalist perspective. It is reader friendly, however, it does not speak to the complexities of de-anonymization from a researches perspective. The material in question appears to be meant for non-technical readers. No attack methods are described in detail and no experiments appear to have been conducted. Ultimately, this paper would not serve the this project as it does not provide or reference any guides to be analyzed and deconstructed.

Design (No Method)

Paper: Exploring and Analyzing the Tor Hidden Services Graph [22]
Impact: Low
Cost: Medium

This paper analyzes topologies of Tor Web graphs. Although it does dabble in the concept of de-anonymization. Its primary focus lies in creating visual illustrations of darkweb topologies and other functionalities.

Signal Tracing Attack

Paper: Tracing Tor Hidden Service Through Protocol Characteristics [23]
Impact: High
Cost: Low

This papers approach lies in compromising adjacent Tor relays to de-anonymize a THS.

In this method, a malicious client establishes a circuit to the THS and sends signals on the circuit. Once the guard of the THS detects the signals, the identity of the THS is disclosed. In layman's terms, Tor clients embed signals into Tor circuits connecting to THSs. When the Tor relay nearest to the THS detects the signal, the identity of the THS can be revealed.

With 100% accuracy, a 99.25% true positive rate, and a 0% false positive rate, this approach appears to be promising.

Exploit RRT Delay

Paper: De-anonymizing Onion Services by Introducing Packet Delay [24]
Impact: High
Cost: Medium

This paper presents an attack method that de-anonymizes THSs by exploiting the delay in Request-Response Time (RRT).

This method involves having the guard of a THS introduce a watermark containing the IP address of the THS in the TCP packet's RRT. The TCP packet transmits the watermark to where the HTTP echo request was sent (from a client to where the RRT was captured and the watermark was decoded). In order to decode the watermark, the normal RRT of the packets on the Tor network was needed. Therefore, in order to get the data, HTTP echo requests were also sent without the watermark.

The paper showcased a success rate of 88.80% where the watermark was decoded by the client. While this method works, the THS needs to choose the guard that introduces the watermark. The chances of that are approximately 0.005%, meaning it would need 20000 tries to break the anonymity of a given THS if only one guard is used.

It is an interesting concept which warrants further exploration.

Keep Tor Circuits Alive

Paper: De-anonymizing Tor in a Stealthy Way [25]
Impact: Medium
Cost: Medium

This paper proposes two methods in which a Tor circuit is kept alive.

The first type is based on protocol-level strategy and attempts to keep connections alive by fixing damaged cells in the network. This adds an additional layer of stealth to the approach.

In the second attack, the guard sends a signal to the accomplice exit node via special types of outbound cells. This prevents the connection from being closed, thus making the attack stealthy for more general situations.

By utilizing these methods, afterward additional analytical approaches can be conducted to de-anonymize the THS.

3.1 Attack Vectors

This section will demonstrate how attack methods derived from the literature review can help establish coherent and precise requirements.

3.1.1 Requirement Specifications

This section aims to establish requirement specifications that coincide with the projects scope and problem statement specified in Section 1.1. These requirements will help bridge the gap between conceptualization and Execution. The goal would be to ensure the attack scenarios are carried out as simply as possible.

Requirements:

The following requirements pertain to the projects outcome and will dictate which attack methods are most appropriate to use to achieve said outcome. In addition, the following requirements will be objectives pertaining to the THS. Every requirement listed has been derived based on the *"Score Description"* table from chapter 3. Ultimately, each requirement has been selected to ensure the chosen attack method would receive the highest possible rating of Impact. In layman's terms, the following requirements must adhere to the definition of *"High Impact"* expressed in the *"Score Description"* table from chapter 3.

- Leak any IP addresses associated to the THS.
- Gain access to the system.
- Escalate privileges while within the system.
- Extract data from the system.

3.1.2 Evaluation

In accordance with the aforementioned literature review and requirements, the following table will evaluate every attack method in contention. Based on their the score of impact and cost, the table will server as a visual illustration that will showcase which attack methods are to be used moving forward.

1st Evaluation			
Method	**Paper**	**Impact**	**Cost**
Fingerprinting	(1)	Low	Medium
Fingerprinting	(2)	Medium	Low
Napping Guard Attack		High	Low
SignalCookie Attack		High	Low
Descriptor Analysis		Low	High
Collection of Attacks	(1)	Low	Medium
CARONTE		Low	Low
Sniper Attack		High	Medium
MitM		Medium	Low
Inflow Technique		High	Low
Exploit Tor v2 Flaw		Low	Low
Collection of Attacks	(2)	High	High
Bitcoin Transfer Analysis		Low	Low
Design (No Method)		Low	Medium
Signal Tracing Attack		High	Low
Journalism (No Method)		Low	Low
Fingerprinting	(3)	Low	Low
Delay RRT Exploit		High	Medium
Circuit Keep Alive Exploit		Medium	Medium
Fingerprinting	(4)	Medium	Low

Finally, as the ideal method of attack would garner a rating of High in impact and Low in cost, the attack methods with this rating are placed in contention.

2nd Evaluation		
Method	**Impact**	**Cost**
Napping Guard Attack	High	Low
SignalCookie Attack	High	Low
Inflow Technique	High	Low
Signal Tracing Attack	High	Low

In order to narrow down the existing selection, additional criteria are considered. For the next selection process time will be considered. This is due to time management being a factor for the project and if the same result can be achieved in less time, that would be an ideal outcome.

3rd Evaluation			
Method	**Impact**	**Cost**	**Time**
Napping Guard Attack	High	Low	High
SignalCookie Attack	High	Low	High
Inflow Technique	High	Low	High
Signal Tracing Attack	High	Low	Low

As an ideal approach emphasizes simplicity, the score most appealing would carry a rating of High in impact, Low in cost and Low in time consumption. Although the SignalCookie attack seemed promising, ultimately, this project will move forward attempting to replicate and potentially improve the *"Signal Tracing Attack"* method. This is also a result of the other attack method taking a more low-level approach.

Infrastructure 4

In this chapter, a step-by-step description of the Tor protocols functionalities will be provided followed by documented work of the THS's design, development and deployment.

4.1 Onion Routing

This section aims to describe how the Tor protocol utilizes multilayered encryption through its relay nodes.

The foundation of Tor was built on onion routing[26]. Onion routers are Tor relays that direct and redirect dark-net traffic. The Tor network is an overlay network that exists on top of the Internet and consists of thousands of these relays. Each relay can be categorized in to one of three categories, that being entry, relay or exit nodes respectively. Where an entry node or *"Guard"* is an initial access point to the Tor network, a relay (or *"middle"*) node relays network traffic as an intermediate and an exit node is the last relay for packets to pass through before reaching their destination. As Tor is open-source, anyone can operate it. Additionally, anyone can operate as an onion router from their desktop computer.

4.1.1 Encryption

Each relay is responsible for a layer of encryption hence the term *"Onion"* routing. The encryption method in question is Advanced Encryption Standard (AES) symmetric encryption. Symmetric encryption means, the same encryption key is used for both encrypting and decrypting messages.

Data is sent through the Tor network in an encrypted format with fixed size packets called *"cells"*. Specifically, encrypted Tor messages are cells that are 512 bytes long. Onion routing is based on the idea that cells can not be discerned based on their features. They constantly remain the same length to decrease the probability of an attacker deducing how many hops in the circuit a cell has taken. In layman's terms, no additional headers are added to a cell that could alter its size. Onion routers differentiate between reading or forwarding cells based on the hash digest residing within the cell. The onion router examines the tail end of a cell, calculates the hash and attempts to match it to its own digest. If the two do not match, the onion router assumes it is still encrypted and forwards the cell to the next relay in the circuit[27].

Figure 4.1 showcases a client that has initialized a Transport Layer Security (TLS) handshake with three separate relays respectively. Each handshake resulted in a shared symmetric session key being generated. The clients packet is encrypted with each key and sent along the circuit. This process is described in detail by figure 4.3.

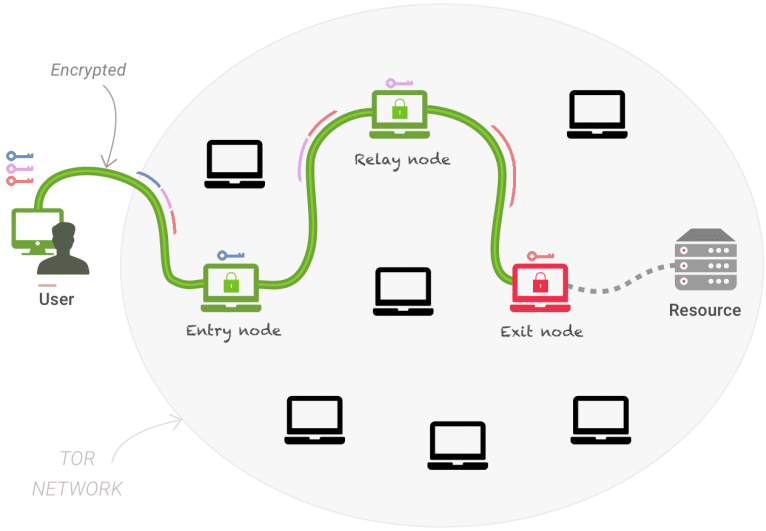

Figure 4.1: Layered encryption[28].

In order to generate these keys, Tor uses the Diffie-Hellman algorithm to establish session keys between clients and onion routers.

By utilizing three separate symmetric keys, the client can compartmentalize network traffic through a series of relay nodes. In other words, a relay node is not previewed to a client or servers information as each relay only possesses information pertaining to the previous and following relay.

Consider a scenario where a client (A) generates a circuit with the relays B, C and D. The cell that the client sends will follow a trail of encapsulation. The process starts with an initial encapsulation with the K_{AD} session key, followed by its encapsulation with the K_{AC} session key and lastly its encapsulation by the K_{AB} session key. The data received by each relay can be described as the following (where M is the original cell):

1) **Data received by *B*:** $K_{AB}(K_{AC}(K_{AD}(M)))$
2) **Data received by *C*:** $K_{AC}(K_{AD}(M))$
3) **Data received by *D*:** $K_{AD}(M)$

When the encrypted cell is received by B from A, relay B would use K_{AB} to decrypt the top layer of encryption (reminiscent to peeling the outermost layer of an onion). At this point it is forwarded to C where the procedure continues until the cell reaches the exit node D, and subsequently the final destination, E.

Transport Layer Security

All connections on Tor use TLS link encryption meaning, the communication between Tor relays is protected by means of a TLS tunnel (e.g. Each relay maintains a TLS connection to every other relay). TLS encryption occurs between the application layer and the transport layer of the TCP/IP model as seen in Figure 4.2. In this layer, application data can be sent securely to the transport layer. Due to the protocol operating between the application and transport layer, TLS can support multiple application layer protocols.

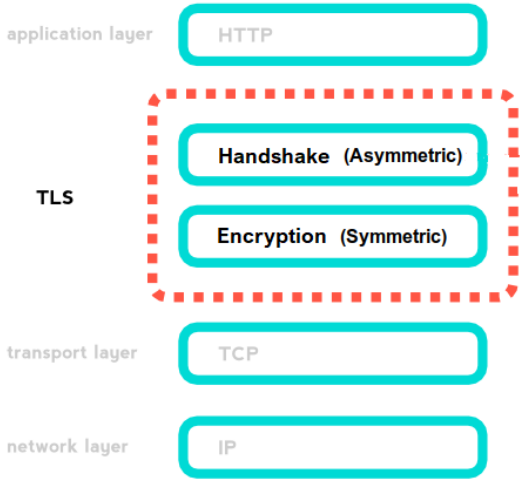

Figure 4.2: TLS Layer

When encrypting, the TLS protocol goes through a few steps. First, the protocol conducts a handshake to initiate the encryption as seen in Figure 4.3. This handshake establishes a set of rules the client and server must adhere to in order to begin secure communications.

The rules in question pertain to those listed in figure 4.3. These rules include cipher specifications (method of encryption e.g. AES etc.), certificate (the signed certificate), key exchange (Diffie-Hellman etc.) and alert. Alert is utilized in cases when errors occur [29]. It should be noted that Tor still uses the outdated TLS v1.2 even though TLS v1.3 was released in 2018.

As depicted in figure 4.3, TLS certificates contain a relays Identity Public Key and are signed by a relays Identity Private Key (based on asymmetric cryptography).

When received, the relay uses the Identity Public Key within the certificate to verify the signature. Every relay on the Tor network has both a Public and Private Identity Key. Although Tor is based on symmetric encryption, TLS uses both asymmetric and symmetric encryption to protect confidential data-in-transit.

Asymmetric encryption is used to establish a secure session between relays, while symmetric encryption is used to exchange data within the secured session. The *"Handshake"* section in Figure 4.2 is where asymmetric encryption occurs (signed by private key & verified by public key) and the session keys are established (symmetric) right before the *"Encryption"* section.

A session key is generated between the client and each relay in a circuit respectively. In other words, the client conducts separate handshakes with each individual relay in the circuit, encrypts the cell with each established session key and then sends it along the chain.

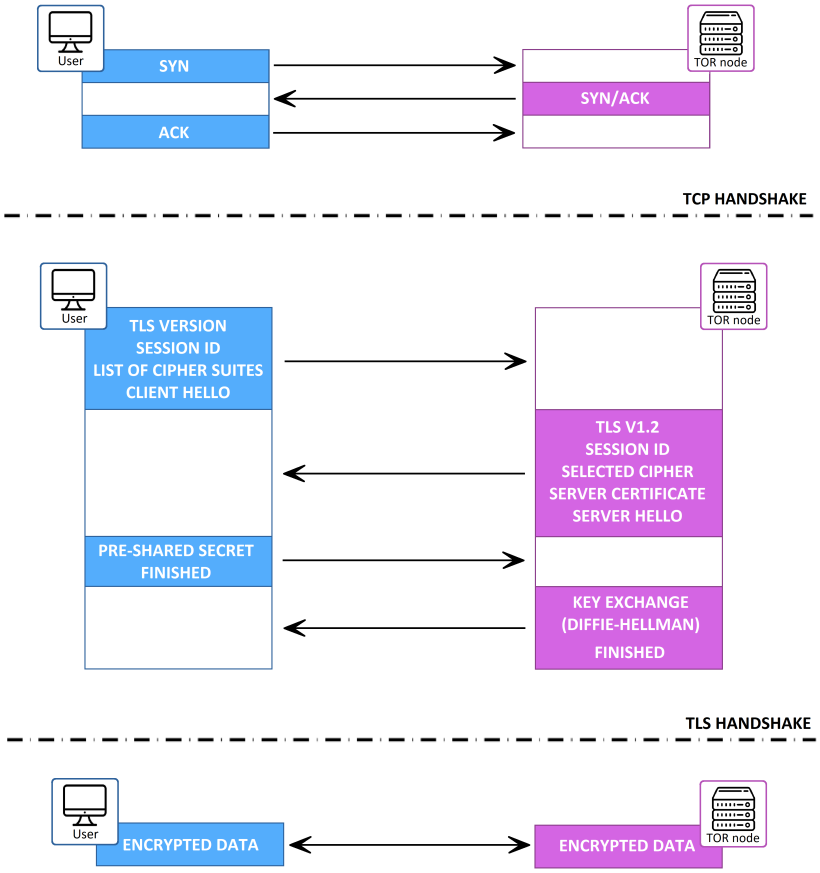

Figure 4.3: TLS Handshake

Additionally, relays typically accept connections on TCP port 443 (HTTPS) since this port is always open on the majority of firewalls. All in all, the Tor protocol anonymizes users by issuing a series of nodes to relay encrypted cells between clients and servers.

4.2 Onion Services

This section will dive into the technological complexities of THS and explain how E2EE is achieved.

Onion services are THS that can only be accessed through a Tor browser. The traffic is encrypted from the client to the onion host. Running a THS gives users all the security of TLS with the added privacy benefits of a Tor Browser. Although all connections in Tor use TLS link encryption, THS use HTTP as opposed to HTTPS as the Tor system already provides E2EE which would make an additional TLS layer redundant.

Additionally, a THS's IP address is protected. THS utilize an overlay network on top of TCP/IP, meaning a client and service's IP addresses are not viewable.

Usually, a client would connect to a server's IP address to access its contents, however, in the case of THS, this cannot be done as the IP address is hidden.

In particular, a THS's address is seemingly a randomly generated string ending in *".onion"*.

The Tor protocol cannot establish a direct connection between a client and service. The client must introduce itself to the service by setting up a rendezvous point (RP) with the service over the Tor network.

The following contains a detailed breakdown of how this occurs.

1) **Introduction Points**

Figure 4.4 depicts the beginning stages of Tor communication between a client and THS. The various components in the illustration include three introduction points, a Distributed Hash-Table (DHT) and various Tor relays. Excluding the client, every component resides within the Tor network (green bubble). The DHT and introduction points are all connected to the THS through relay nodes.

To establish contact between a client and service, various steps must be followed in accordance with the Tor protocol. The first step in the protocol pertains to the service in question. The service will contact a series of Tor relays and request they act as its introduction points as seen in figure 4.4. It does this by establishing long-term circuits to them. A circuit refers to a path of Tor relays, network-traffic travels along. The service connects to the introduction points through a two-hop Tor circuit. As circuits are anonymized, at no point is the service location revealed to the introduction points.

Furthermore, the service will receive additional protection by only allowing contact through the three introduction points.

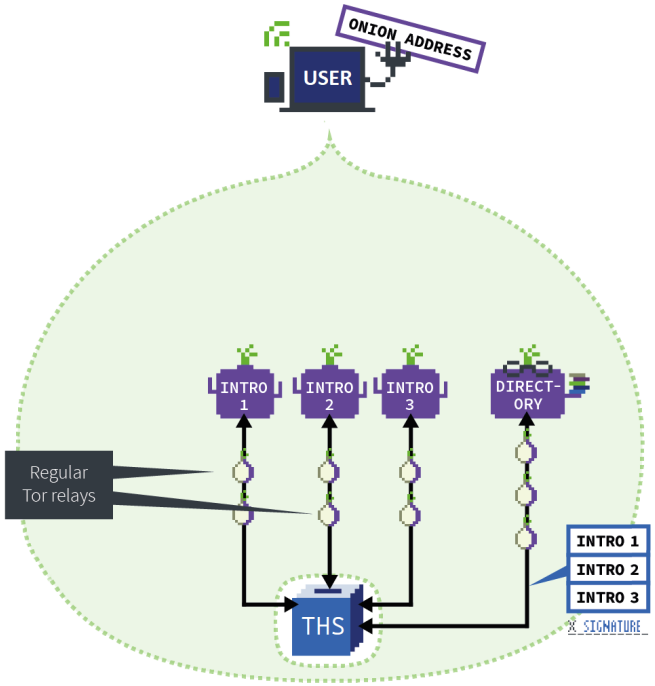

Figure 4.4: Introduction Points [30].

At this point the question becomes, how will clients be able to find these introduction points? Tor provides additional features which allow services to assemble a service descriptor. The service descriptor contains a list of IP addresses (from the service's introduction points) and the service's Identity Public Key. This descriptor is signed with

the service's Identity Private Key as seen in Figure 4.4. This information is encoded in the service's onion address.

Figure 4.4 shows, the service uploading the signed descriptor to a DHT (elaborated in the next step). The DHT is part of the Tor network which makes it readily available for clients. The upload is done using an anonymized Tor circuit which ensures the service location is not revealed.

2) **DHT**

Figure 4.5 depicts the secondary stage of dark-net communication between the client and THS. The introduction points are all connected to the THS through relay nodes. The DHT, however, is connected to both the client and service respectively. Furthermore, the figure showcases the passing of data from the DHT to the client. The data in question pertains to the THS descriptor.

The previous step was set-up for the service so it could be accessed by clients.

For a client to view a service's content, a Tor browser and an onion address are needed. In this case, the client goes to the DHT and requests the signed descriptor of the service as seen in Figure 4.5. The DHT provides a signed list of all the known relays, and in that list are a set of certificates from each relay (self-signed by their Identity Key) specifying their keys, locations, exit policies, and so on.

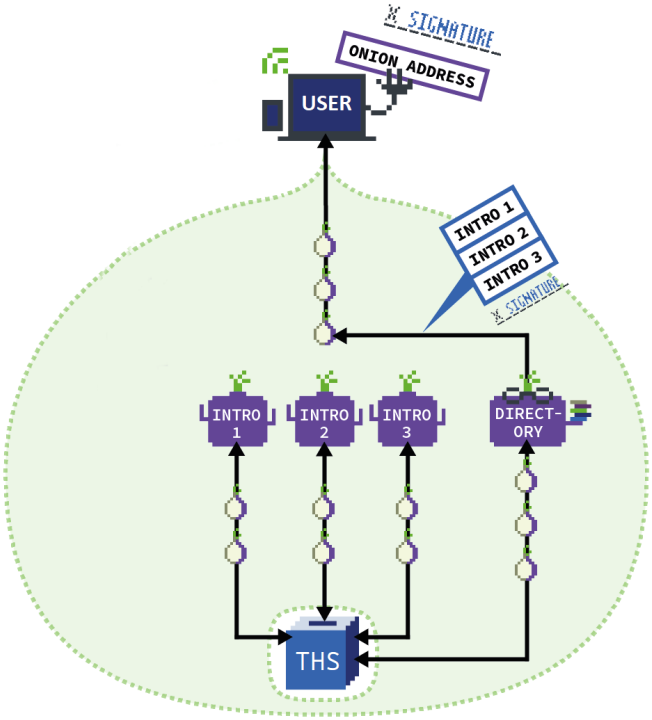

Figure 4.5: Distributed Hash-table [30].

When the client receives the signed descriptor, the descriptors signature is verified using the Identity Public Key that is encoded in the onion address (figure 4.5). This ultimately provides E2E authentication as the descriptor could only have been produced by that service and no one else.

Now that the client has obtained the service's authentic descriptor, the IP addresses of the various introduction points can be extracted from them.

3) **Rendezvous Point**

Figure 4.6 depicts the third stage of dark-net communication between a client and THS. The various components in the illustration include three introduction points, a RP and various Tor relays. As previously stated, the introduction points are all connected to the THS through relay nodes. The RP, however, is connected to the client alone. Furthermore, the figure showcases the passing of data from the client to the RP and an introduction point. The data in question partially pertains to the RP. The other portion of the data is a shared secret. The figure additionally shows the service handling the data in question after receiving it.

Before any data transfers can take place, the client must establish a circuit to a chosen Tor relay and request it become their RP. Following that, the client passes a one-time shared secret to the rendezvous and introduction points respectively. This will be essential during the rendezvous procedure (Figure 4.6).

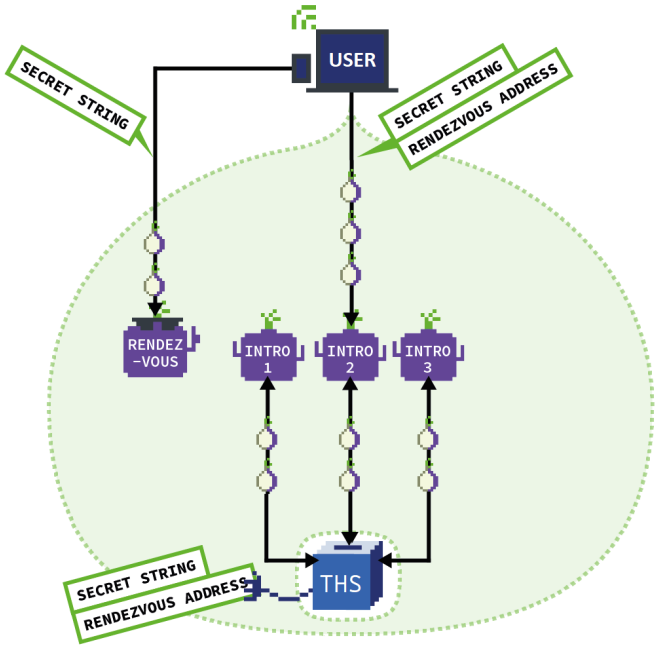

Figure 4.6: Rendezvous Point [30].

From the introduction point, the shared secret and rendezvous address are passed on to the service. Here, multiple verification processes are ran to decide whether the source is trustworthy or not as seen in figure 4.6.

4) **Authentication Token**

Figure 4.7 depicts the fourth and final stage of dark-net communication between a client and THS. This time the RP, is connected to the client and service respectively. Additionally, the figure showcases the passing of data from the client to the RP and the service to the RP. The data in question pertains to the aforementioned shared secret.

Once the validity of the RP has been confirmed, the service establishes an anonymized circuit and connects to the RP. After sending the shared secret to the RP, one final verification is done. This is done by comparing the secret strings from both the client and service respectively (the latter is also from the client but has been relayed through the service) as seen in figure 4.7. In this instance, the shared secret from the client and service operate as authentication tokens.

Finally, with an established secure connection, the RP can simply operate as an intermediate relay where E2EE cells are sent from client to service and vice versa.

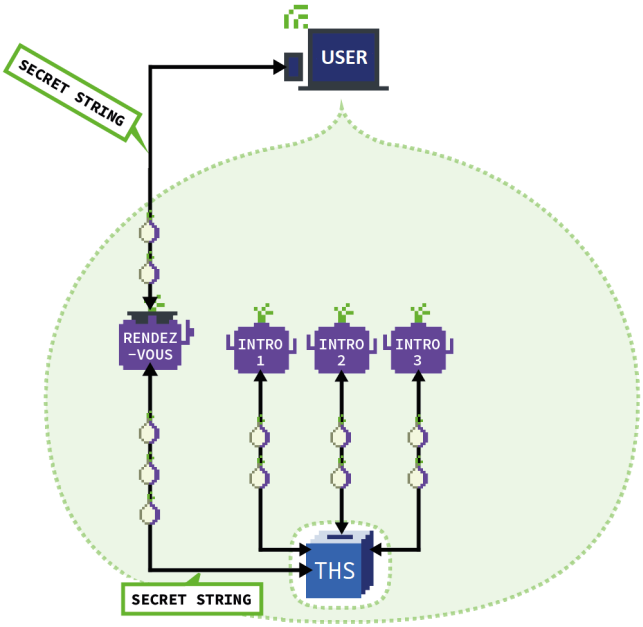

Figure 4.7: Authentication Token [30].

In conclusion, the completed connection between client and service consists of six relays. Three out of the six relays were chosen by the client with the third being the RP and the other three were chosen by the service resulting in secure location hiding for both parties.

4.3 Application Architecture

This section describes the various criteria the THS must adhere to. Additionally, a descriptive reenactment of the THS design, development and deployment will be described. The THS in question had already been developed for a previous paper[31], therefore, most of the same software is utilized.

4.3.1 Requirement Specifications

This section aims to establish requirement specifications that coincide with the projects scope and problem statement specified in Section 1.1. These requirements will narrow the gap between the conceptualization and implementation phase. The goal would be to ensure the implementations simplicity and coherency.

Requirements:

- The THS must have all features of an application service hosted on the dark-net.
- The THS must have a login system for various users to insert data.
- The THS must handle user data with a database.

4.3.2 Design

This sections aim is to give a detailed description of the THS flow from a user perspective.

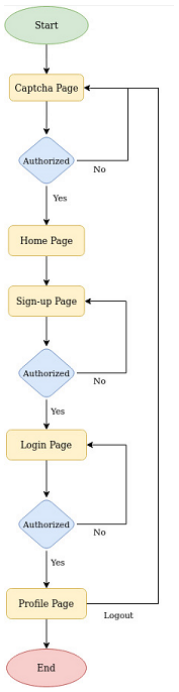

Figure 4.8: Flowchart

The THS is a dark-net web application with basic front-end and back-end features. The application is a basic login system with captcha, login, registration, home and user profile pages respectively. It provides basic security functionality relative to the offensive demonstration that is to take place. In accordance with the aforementioned requirements, the application must have features that would make information disclosure a possibility e.g, database storage for user information. The flow in which a user would interact with the application can be seen in Figure 4.8.

A sub-requirement for the application was to provide an attacker with a storage element that could be exploited as mentioned in 3.1. Due to that fact, the application has basic security features as to add to the authenticity of the demonstration. These security features include hashed passwords and session cookies to avoid URL bypassing.

An additional security feature reminiscent to that of a real world THS is the captcha page. Captcha pages are used in THSs to avoid bot activity. In the case of a THS, captcha pages are usually complex and dynamic, however, for the purposes of this demonstration, an overly complex captcha page was deemed unnecessary. Therefore, a standard text based captcha was used.

In this case, a new random string is generate any time a user inserts an incorrect string or the page reloads. Naturally, the only way to bypass the captcha page is to insert the correct string.

4.3.3 System Components

This section will describe the inner workings of the application and its architecture.

The application was structured with a monolithic architecture as opposed to a cloud native approach utilizing micro-services. It consists of client-side and server-side functionalities. The server-side components include an Application Programming Interface (API) and a database (Figure 4.9), whereas the client-side is front-end HTML.

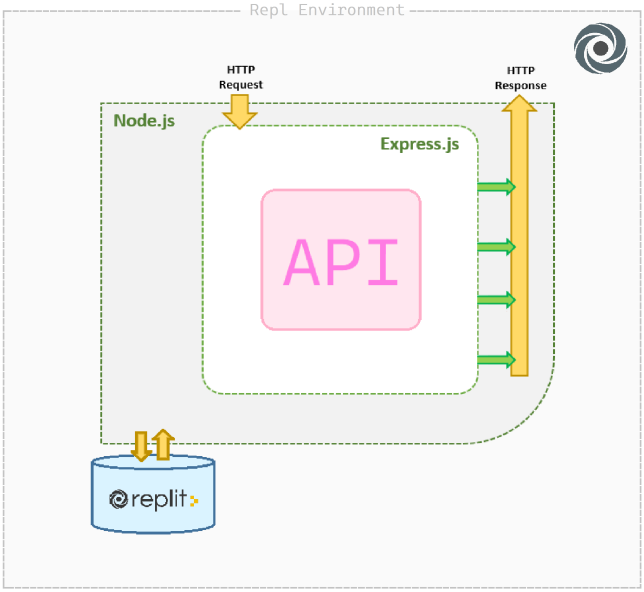

Figure 4.9: Architecture[32]

The client-side manages all front-end activity, and provides users with a Graphical User Interface (GUI). Coded in HTML and CSS, it handles anything a user would interact with while on the site. Put simply, it is responsible for the applications visual aesthetics.

Unlike the front-end that controls everything the user can see, the back-end is involved in data storage, query handling, authentication security, and other server-side functions hidden from the user.

The API is responsible for the back-end functionality. Figure 4.9 mainly illustrates the virtualized back-end components. The term virtualized referring to the fact that the entire service operates within a Docker container. By virtualizing the application, circuit analysis and fingerprinting can occur within a controlled environment. Additionally, it makes the application easier to migrate to new hardware.

Implemented with *"Nodejs"* and *"Express"*, the API links the database to the front-end. It does this by managing all HTTP requests between the two.

Finally, the database resides in the back-end of the application along side the API. The application utilizes a Replit database service and has a key-value based data storage infrastructure.

4.3.4 Deployment

In this section, a clear depiction of the deployment methods used will be displayed.

Hidden Service

By forking an existing dark web repository[33] and making slight modifications, the application could be deployed in a remote location. The following shows the applications file structure.

```
./
├── domain.sh
├── server.js
├── .torrc
├── .replit
├── node_modules
├── package.json
├── packagelock.json
├── public
│   ├── pic.jpg
│   ├── pic1.jpg
│   ├── user.jpg
│   └── style.css
├── views
│   ├── captcha.ejs
│   ├── index.ejs
│   ├── login.ejs
│   ├── register.ejs
│   └── profile.ejs
├── README.md
├── replit.nix
├── run.sh
└── tor
    └── hidden_service
        ├── authorized_clients
        ├── hostname
        ├── hs_ed25519_public_key
        └── hs_ed25519_secret_key
```

The platform the application was deployed on was the online IDE, Replit[34]. Replit allows developers to create projects and write code online.

In the aforementioned file structure, the most important files were the following:

- **run.sh**
  This is the bash script that runs *"server.js"*. This ultimately sets up everything and starts the Tor service.
- **domain.sh**
  Tor automatically generates a *".onion"* domain when the site runs. The domain is a random ‘56‘ character long string that can be customized. It is stored in the *"tor/hidden-service/hostname"* file which can be extracted by running the *"domian.sh"* script.

- **.torrc**
  This file contains all general configurations and equates to *"/etc/tor/torrc"* in other linux platforms.
- **tor folder**
  The folder that is the container for the THS. (equivalent to /var/lib/tor/ in other linux platforms)
- **.replit**
  This is the Replit configuration file and Configures things like the run button etc.
- **.replit.nix**
  The nix file contains the packages the service needs to run on Replit.

The specific configurations in *".torrc"* are as follows:

```
1 HiddenServiceDir ./tor/hidden_service/
2 HiddenServicePort 80 127.0.0.1:80
3 EntryNode $fingerprint
```

Snippet 4.1: THS torrc Configurations

Entry Node

The entry node was each deployed on a Raspberry Pi. The Raspberry Pi would operate as a remote server for the node as opposed to it being hosting with Replit's cloud services. This decision was based on the fact that, the Rasberry Pi offers more processing power in addition to a variety of features Replit does not. However useful Replit is, it only offers a standard Linux based CLI for development purposes.

In order to make the Raspberry Pi operate as a Tor relay, a few steps had to be taken:

1) **Install Tor:**

```
1 $ sudo apt update
2 $ sudo apt install tor
```

Snippet 4.2: Tor installation

2) **Edit the *torrc* configuration file:**

```
1 $ sudo nano /etc/tor/torrc
```

Snippet 4.3: Edit configuration file

Add the following to the guards *torrc*:

```
ControlPort 9051
CookieAuthentication 1
ORPort      9001
Nickname    aauentry
```

Snippet 4.4: Guard torrc Configurations

3) **Restart Tor with the new changes added:**

```
$ sudo service tor restart
```

Snippet 4.5: Tor Restart

Now that the THS's guard has been deployed, a Proof-of-Concept (PoC) of the *"Signal Tracing"* attack method can be conducted in the following chapter as described by Figure 4.10 and chapter 3.

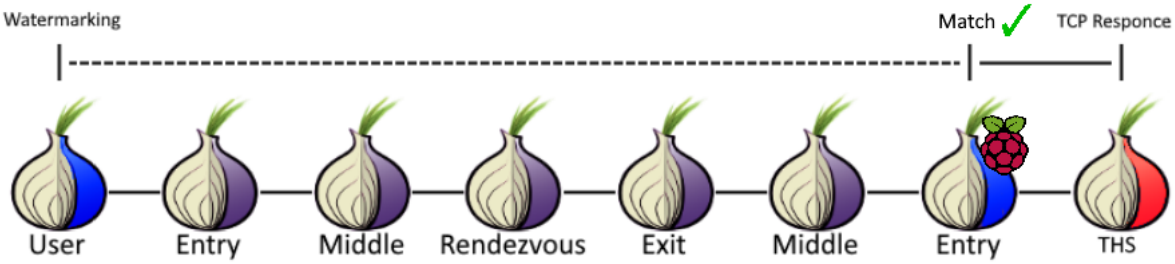

Figure 4.10: Raspberry Pi Circuit[24]

De-anonymization 5

This chapter showcases a PoC of the *"Signal Tracing"* attack method and culminates in an assessment of the findings.

5.1 Conceptualization

This section will describe the procedures and difficulties an attacker must be mindful of in order to de-anonymize a THS.

As described in Chapter 3, the *"Signal Tracing"* attack method relies on an attacker compromizing the THS's guard. This is advantageous as the malicious guard is the only relay in the circuit capable of divulging the IP address of the THS. This is due to its direct line of communication to the THS.

Once a malicious guard has been established, the next step would be to ensure the THS prioritizes the compromised relay and connects to it. This, however, is unlikely as Tor relays doe not associate themselves unilaterally to singular Tor entities.

The initial idea was to specify a unique payload for an HTTP request (can be done through a browsers ***"Inspect"*** feature) at which point, a monitoring tool running on the malicious guard (akin to Wireshark) could filter the traffic for the aforementioned payload. By doing so and by attributing the destination IP address to that of the THS, the successful IP leakage of the THS would be the result.

Although this method is simplistic, there are alternate ways to match signals that could potentially also be explored. One of which was specified in Chapter 3, where the authors of the *"Signal Tracing"* attack method utilized complex algorithms in order to match signals at the compromised guard.

Other methods include *"Exploit RRT Delay"* (Chapter 3) and other watermarking techniques. A digital watermark is a form of information embedded within digital content that is used to identify its origin, ownership, or authenticity. One can embed a signal by adding watermarks to packets in HTTP requests.

Furthermore, methods to increase a relays *"trust"* could be explored. By increasing the *"trust"* of the malicious guard, adjacent Tor entities may associate themselves with it. Although not a guarantee, this would be one way of making the target latch itself to the compromised relay.

One method of increasing *"trust"* would be by simulating relay criteria. Essentially, this would require spoofing a relays data to raise its consensus weight.

Other ways of completing these objectives will be explored throughout the process.

5.2 Scoping & Requirements

The aim of this section is to express the scope and parameters the de-anonymization process must adhere to.

After conducting extensive research on existing attack methods to de-anonymize THSs, it has become evident that the methods described are entirely too theoretical. Although most of the papers mentioned in Chapter 3 were in-depth in terms of the statistics and mathematics involved, they gave no mention to the tools used or anything akin to a step by step reenactment of how these experiments were conducted, practically.

Therefore, this chapters scope will be centered around one question:

- How does one practically expose a THS's IP address?

Additionally, the requirements have been selected to ensure the attack method in question would be carried out as efficiently as possible.

- Launch a compromise guard.
- Connect the compromised guard to the THS.
- Monitor in-going and out-going traffic on the compromised relay.
- Specify a unique payload in the attackers HTTP requests.
- Filter traffic for the unique payload on the compromised relay.
- Increase the guards perceived validity.

5.3 Design & Implementation

This section will attempt to showcase a PoC of Hidden Service IP leakage and the tools required to achieve that objective.

5.3.1 Connecting to Guard

Before testing can occur, initial set-up is required. As previously mentioned, the THS must also configure its *torrc* file to the following:

```
1 EntryNode $FINGERPRINT
```

Snippet 5.1: THS torrc Conf

This will ensure the THS utilizes a specific guard when establishing circuits. However, in an attempt to connect the service to the guard, an error occured (depicted in Figure 5.1). This error describes the service's attempt to utilize Tors relay allocation features as a potential security breach and thus will not allow the user to proceed.

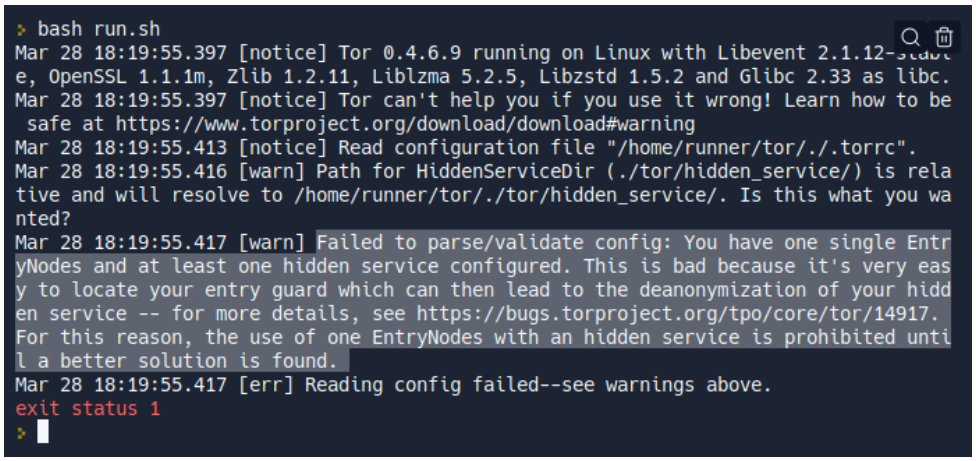

Figure 5.1: Connection Error

Consider a scenario where an attacker selects a rendezvous point which happens to be the THS's primary guard[35]. The THS would look through its preemptively built circuits and evaluate whether any of them are extendable. Tor functions would state that none of the existing circuits are suitable. This is the case as any potential circuit would cause a duplicate hop. In addition to being a functionality issue, it also represents a security risk as an attacking clients choice of rendezvous point could potentially leak information about a THS's guard and thus the service itself[35].

Alternative Sources

In order to solve this issue, a variety of options were considered. First, the original authors of the *"Signal Tracing"*[23] paper were contacted. However, the email addresses specified in the paper were seemingly no long active as the mails sent were cancelled.

At this point, ChatGPT was consulted. ChatGPT is an artificial-intelligence (AI) chatbot developed by OpenAI and launched in November 2022. It utilizes large language models and

has been fine-tuned using both supervised and reinforcement learning techniques. By simply writing a question, the AI chatbot provides a response within seconds. It is a useful source of information (akin to Google.com) and is applicable in most fields. The following question was asked:

- how do you make a hidden service connect to a specific guard?

The chatbot provided the following response:

> *"When setting up a Tor hidden service, it is not possible to choose a specific guard. The Tor network uses a distributed and decentralized system where the choice of guard is made by the Tor client software."*

This initial set back required the project take a different approach. Luckily, the AI provided a few suggestions:

> *"However, there are some ways to increase the likelihood of connecting to a specific guard, such as:* ***Using a Tor bridge*** *or* ***Using the Tor Control Protocol****."*

These options can be described as follows:

- **Tor bridge:** A Tor bridge is a private entry point that is not listed in the public Tor directory. Bridges are ordinarily utilized so ISPs or governments trying to block access to the Tor network can't. They are especially useful when users do not want to be noticed contacting a public Tor relays IP address. In this instance, the bridge would be used as the primary guard of the THS.
- **Tor Control Protocol:** The Tor Control Protocol allows users to interact with the Tor client software and configure certain aspects of its behavior, including which guard to use. For this scenario the user would have to write a script running the Tor Control Protocol to select a specific guard based on its fingerprint.

5.3.2 Tor Bridge Test

This project will attempt to move forward with the Tor bridge alternative. The decision was based on the simplicity involved when converting an ordinary guard to a bridge. This ultimately runs in contrast to the complexity required to develop a script utilizing the Tor Control Protocol.

In order to make the Raspberry Pis operate as Tor bridges, a few steps had to be taken:

1) **Install Tor:**

```
$ sudo apt update
$ sudo apt install tor
```

Snippet 5.2: Tor installation

2) **Install the latest version of *obfs4proxy*:**

```
$ sudo apt install obfs4proxy
```

Snippet 5.3: Install obfs4proxy

3) **Edit the *torrc* configuration file:**

```
$ sudo nano /etc/tor/torrc
```

Snippet 5.4: Edit configuration file

Add the following to the guards *torrc*:

```
BridgeRelay 1
ORPort      5000
ServerTransportPlugin obfs4 exec /usr/bin/obfs4proxy
ServerTransportListenAddr obfs4 0.0.0.0:6000
ExtORPort auto
CookieAuthentication 1
Nickname    aauentry
```

Snippet 5.5: Bridge torrc Configurations

4) **Restart Tor with the new changes added:**

```
$ sudo service tor restart
```

Snippet 5.6: Restart Tor

5) **Check the bridge logs:**

```
$ journalctl -e -u tor@default
```

Snippet 5.7: Log Check

6) **Finally, specify the bridges fingerprint in the THS's configuration file:**

```
EntryNode $FINGERPRINT
```

Snippet 5.8: THS torrc Configurations

Now that the Bridge has been configured, the next steps can be taken.

Troubleshooting

By checking the logs, the guard was successfully turned into a private Tor bridge. Unfortunately, the connection errors persisted. Finally, to ensure this was the right path, Tor developers responsible for writing the error message were contacted. The following messages were sent:

> *"Hello,*
>
> *I'm a Masters student writing my thesis on de-anonymization. I've been attempting to make my hidden service use a specific guard, however, an error occurs which pointed me to this thread. While looking into options, I discovered that by making my guard a Tor bridge, I might be able to bypass these security features. This didn't work and ultimately sent me back to this thread. I was wondering if there was a tangible way to make my hidden service use my guard. Whether it be by editing the torrc file or by injecting the relays fingerprint in the hidden service directory. Specifics would be nice, thanks in advance."*

Unfortunately, no maintainer responded to the emails nor the message left in the Gitlab discussion thread.

It was later discovered that this error could be avoided by adding additional guard options to the THS's *"torrc"* configuration file. This method attempts to fix the aforementioned *"RP=Guard"* problem. Although it is not a perfect option, it does provide a protective layer through redundancy.

Through further experimentation, it became apparent this *"security feature"* was not as well implemented as originally assumed. Not only could the previously mentioned error be avoided by adding additional guard options, it could be bypassed by inserting the same guard ID (or fingerprint) twice. Either of the following commands have this effect:

```
1 EntryNodes $FINGERPRINT,$FINGERPRINT
2 EntryNodes $FINGERPRINT1,$FINGERPRINT2
```

Snippet 5.9: THS torrc Configurations

By using the first command, the THS infrastructure would appear as seen in figure 5.2. In this case, the THS would use the same guard for each established circuit. This would mean, the THS would connect to the RP and introduction points, respectively via the same guard.

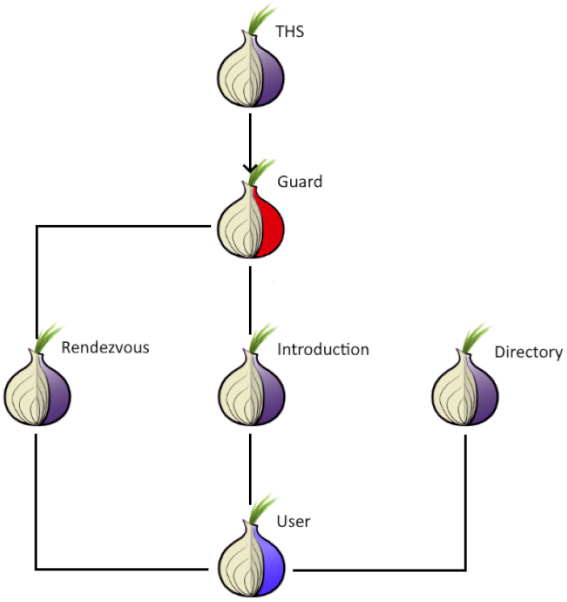

Figure 5.2: THS connecting from a single guard

While resolving the previous issue, this method paved the way for a new error to present itself. Figure 5.3 shows the new issue:

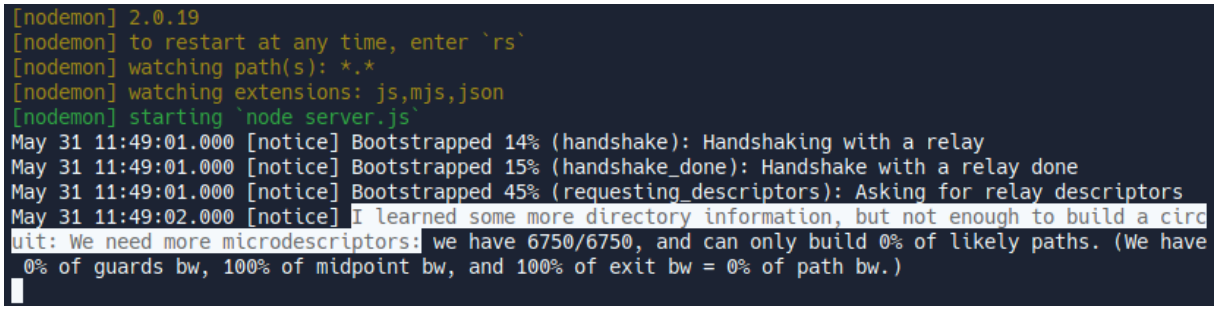

Figure 5.3: Missing Microdescriptors

The message reads *"I learned some more directory information, but not enough to build a circuit: We need more microdescriptors:"*

This could be due to a number of reasons.

- **Tor Bridge vs Traditional Guard:** As the current guards in use are set up as Tor bridges, there may be some configuration prerequisite required to make things function. Therefore, the same experiment was conducted where two guards were deployed on replit and their fingerprints were added to the THS *torrc* file. This made no difference, unfortunately.
- **Firewall issue:** A Firewalls configurations could be interfering with the packets being sent.

5.3.3 Firewall Checks

The first step in troubleshooting these issues is by conducting firewall checks. To do this, the *"iptables"* settings will be configured. *Iptables* is a firewall utility for Linux-based operating systems[36]. It is used to manage network traffic by setting up rules and policies for packet filtering. Unlike *"UFW"* (Uncomplicated Firewall), *"iptables"* operates at the kernel level, allowing it to examine and manipulate network packets as they pass through[36].

Iptables must be configured to allow Tor traffic to pass through the relay. To do this appropriate rules to permit incoming and outgoing connections related to Tor must be created. The following settings reveal how this is done:

1) **Identify Tor ports:** By default, Tor uses TCP port 9001 for incoming connections and TCP port 9030 for directory connections. Therefore, these ports were first specified in *torrc*.
2) **Allow incoming Tor traffic:** The following commands are used to create rules that allow incoming connections on the Tor ports.

```
1 $ sudo iptables -A INPUT -p tcp --dport 9001 -j ACCEPT
2 $ sudo iptables -A INPUT -p tcp --dport 9030 -j ACCEPT
```

Snippet 5.10: Allow incoming

These rules specifically allow incoming TCP connections on ports 9001 and 9030.

3) **Allow outgoing Tor traffic:** The following commands are used to create rules that allow outgoing connections from the Tor ports.

```
$ sudo iptables -A OUTPUT -p tcp --dport 9001 -j ACCEPT
$ sudo iptables -A OUTPUT -p tcp --dport 9030 -j ACCEPT
```

Snippet 5.11: Allow outgoing

These rules specifically allow outgoing TCP connections on ports 9001 and 9030.

4) **Set the default policy:** The following commands are used to ensure that the default policy for incoming, outgoing, and forwarded packets are set to ACCEPT. This ensures that if no specific rule matches, the packet is allowed to pass through.

```
$ sudo iptables -P INPUT ACCEPT
$ sudo iptables -P OUTPUT ACCEPT
$ sudo iptables -P FORWARD ACCEPT
```

Snippet 5.12: Set default policy

5) **Save rules:** In order to ensure the *iptables* rules stay update regardless of reboots, they need to be saved to a specific file.

```
$ sudo iptables-save > /etc/iptables/rules.v4
```

Snippet 5.13: Save changes

Unfortunately, the error persisted, meaning, new methods of troubleshooting had to be considered.

Port Scans

As the ORPort (Onion Router Port: 9001) is the port a relay uses to connect to the Tor network, a logical next step would be to check if these ports were open. This can done with tools like *Netcat* or *Nmap*.

To check if the ORPort is open using nmap, a port scan on the target IP address can be used to determine if the specified port is open or closed:

```
$ nmap -p <ORPort> <IP>
```

Snippet 5.14: Nmap scan

Additionally, to check if the ORPort is open using netcat, a simple TCP connection test can be used to the specified port.

```
$ nc -vz <IP> <ORPort>
```

Snippet 5.15: Netcat test

Interestingly, the port appeared to be open initially. This resulted in the following response from the *nmap* port scan:

```
    $ nmap -p 9001 192.168.0.159
Starting Nmap 7.80 ( https://nmap.org ) at 2023-05-14 15:18 CEST
Nmap scan report for 192-168-0-159.ip.linodeusercontent.com
   (192.168.0.159)
Host is up (0.061s latency).

PORT     STATE SERVICE
9001/tcp open  tor-orport

Nmap done: 1 IP address (1 host up) scanned in 0.36 seconds
```

Snippet 5.16: Netcat test

The following are the results from the *ncat* test:

```
    $ nc -vz 192.168.0.159 9001
Connection to 192.168.0.159 9001 port [tcp/*] succeeded!
```

Snippet 5.17: Netcat test

It became apparent that these results only occurred when the raspberry pi's private IP address was used. When conducting the same tests using the public IP address, the port was shown to be closed or filtered. This could only mean there was a firewall configured in the router ensuring no Tor traffic from port 9001 could be retrieved (on this particular LAN (Local Area Network)).

This revelation made three options available.

- Reconfigure Routers Firewall
- Change ORPort (and possibly others)
- Deploy on cloud service

Reconfiguring the router is not an option as the current one in use is the property of multiple residents. A new router could be configured for Tor dependencies, however, at this point in the project exploring this new avenue would not be ideal due to time constraints.

Furthermore, changing default Tor ports could result in an entirely new subset of problems relating to compatibility. Therefore, Linode's services were used.

5.3.4 Linode Deployment

Linode is an IaaS (Infrastructure-as-a-Service) provider who offers virtualized computing resources over the Internet[37]. These resources, include VMs (Virtual Machines), storage, and networking infrastructure, to build and deploy applications and services[37]. As an IaaS, Linode would not provide any pre-installed firewall configurations that could tamper with the relay.

A Debian 11 instance was deployed and configured using the following steps:

1) **Ensure the newest version of Tor is installed:** Tor needs to be installed this way due to Debian 11 installing older versions of Tor when using *"apt"*. The versions shown to be available with ***"apt-cache madison tor"*** are considered obsolete and would effect the relay's consensus weight.

```
$ sudo apt update
$ wget https://dist.torproject.org/tor-0.4.7.13.tar.gz
$ tar -xzf tor-0.4.7.13.tar.gz
$ cd tor-0.4.7.13/
```

Snippet 5.18: Download Tor

2) **Install missing dependencies:** Running the binary on the *"configure"* file may be unsuccessful due to missing dependencies. Therefore, the following installations are made:

```
$ sudo apt install build-essential libevent-dev libssl-dev zlib1g-
    dev
```

Snippet 5.19: Install missing dependencies

3) **Run binary on file named *"configure"*:**

```
$ ./configure
```

Snippet 5.20: Run binary

4) **Finalize installation by running make commands:**

```
$ sudo make
$ sudo make install
```

Snippet 5.21: Installation

5) **Create torrc file:** Running the ***"tor"*** command may not work as it searches for the *torrc* file when running. After running ***"make install"*** a tor configuration file is generate, however, it may have a different name e.g. *torrc.sample*. Therefore, after making the proper changes to *torrc.sample*, it should be moved to a file in the specified directory path with the correct name.

```
$ sudo mv /usr/local/etc/tor/torrc.sample /usr/local/etc/tor/torrc
```

Snippet 5.22: Create torrc

6) **Run Tor command to start Tor:**

```
$ tor
```

Snippet 5.23: Run Tor

Performance Checks

The following checks are to ensure everything is running correctly:

1) **Network checks:** The following *netstat* command is used to list open ports and listening interfaces related to Tor.

```
1     $ sudo netstat -plntu | grep tor
2 tcp        0      0 127.0.0.1:9050          0.0.0.0:*
                    LISTEN      522/tor
3 tcp        0      0 127.0.0.1:9051          0.0.0.0:*
                    LISTEN      522/tor
4 tcp        0      0 0.0.0.0:9030            0.0.0.0:*
                    LISTEN      522/tor
5 tcp        0      0 192.46.236.98:9001      0.0.0.0:*
                    LISTEN
```

Snippet 5.24: Listening interfaces

Alternatively, the *"lsof"* command could be used. It lists open network connections and filters the output to display only the lines related to Tor.

```
1     $ sudo lsof -i -P -n | grep tor
2 tor        522    root 6392u  IPv4 729654      0t0  TCP
     192.46.236.98:36962->79.211.46.157:9001 (ESTABLISHED)
3 tor        522    root 6393u  IPv4 430474      0t0  TCP
     192.46.236.98:38674->95.128.43.164:443 (ESTABLISHED)
4                          .
5                          .
6                          .
7                          .
8 tor        522    root 6858u  IPv4 840348      0t0  TCP
     192.46.236.98:9001->57.128.83.98:36918 (ESTABLISHED)
9 tor        522    root 6864u  IPv4 839630      0t0  TCP
     192.46.236.98:9001->194.104.156.9:21798 (ESTABLISHED)
```

Snippet 5.25: Network connections

2) **Solve potential port issue:** An additional error that could occur when running a Tor relay is TCP port exhaustion[38]. TCP port exhaustion refers to a situation where all available ports on a device are in use or unavailable for new connections. It primarily occurs when a large number of connections are being made to the device simultaneously, exceeding its capacity to handle incoming connections. This situation can lead to various issues, including the inability to establish new connections, dropped or rejected connections, and degraded performance.

   The issue can be resolved by tuning sysctl (System Control).

   By tuning sysctl one can expand the devices local port range. This can be done with the following command:

```
1 $ sudo sysctl -w net.ipv4.ip_local_port_range="15000 64000"
```

Snippet 5.26: Tuning sysctl

It should be noted that tuning sysctl as described is not permanent and will be lost upon restart. In order to make this change permanent, the changes need to be added to the configuration file */etc/sysctl.conf*:

```
$ sudo nano /etc/sysctl.conf
```

Snippet 5.27: Edit sysctl configuration file

The following line should be added to the end of the file:

```
net.ipv4.ip_local_port_range="15000 64000"
```

Snippet 5.28: Tuning sysctl perminantly

Final *torrc* Configurations

Now that the correct installation method has been established, a method to determine which relay configurations would result in the highest possible consensus weight, must be defined.

In order to determine a baseline of what high priority Tor relays had in common, the Tor metrics website[39] was used. Tor metrics enables users to analyze live anonymity systems. For security purposes, Tor metrics ensures that data analysis is performed with great care so that the users' privacy is not put at risk.

A search was done of the top relays possessing the highest consensus weights and the *torrc* file was configured to replicate their features:

```
$ sudo nano /usr/local/etc/tor/torrc
```

Snippet 5.29: torrc edit

The following configurations were set:

```
ORPort <public_IP>:9001
DirPort 9030
ControlPort 9051
HashedControlPassword <hashed_password>
ContactInfo <email>
RelayBandwidthRate <number>
RelayBandwidthBurst <number>
ExitPolicy reject *:*
Nickname <your_nickname>
```

Snippet 5.30: torrc settings

To calculate the Tor relay's ***RelayBandwidthRate*** and ***RelayBandwidthBurst*** one must consider the available network bandwidth for outbound and inbound connectivity.

Determining the maximum network bandwidth available can be done by consulting ones ISP (Internet Service Provider) or network administrator. In this case, the website *Public Cloud Reference*[40] by Cloud Mercato was referenced. Public Cloud Reference is a website that collects and analyzes data from cloud providers.

The Debian 11 Linode instance used for this paper was a 4G RAM plan for $36 monthly. As listed in the Public Cloud Reference, this particular plan has the following network bandwidth data[40]:

Network Bandwidth	
	Bandwidth
Max	3940
Average	2874
Deviation	957.64
Min	1105

These statistics can be further illustrated by the following graph:

- Green is the upload speed
- Red is the download speed

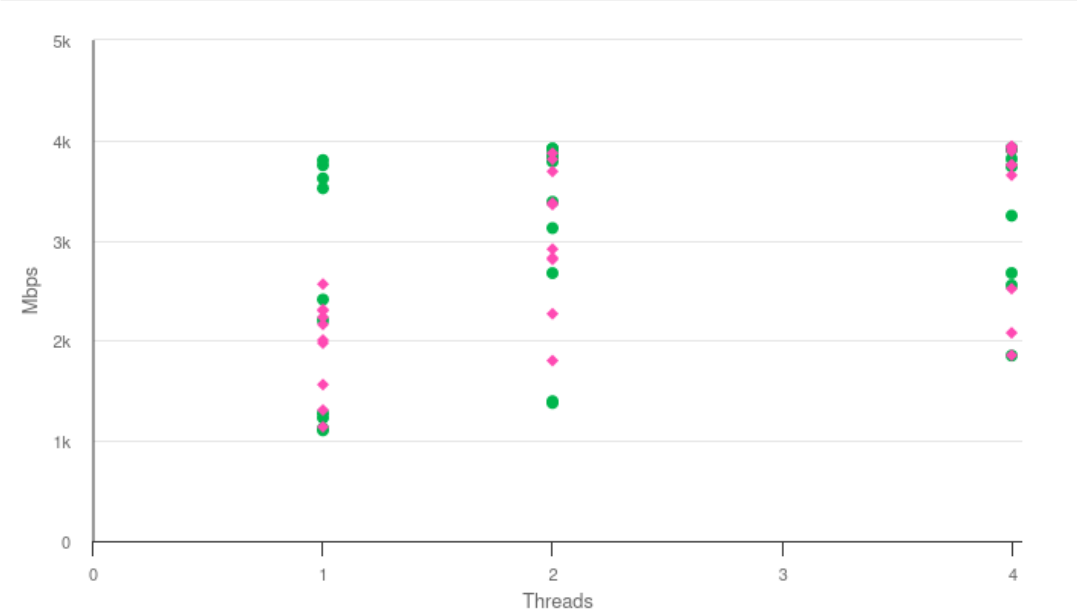

Figure 5.4: Network Bandwidth Statistics[40]

Given the above information, you can estimate the RelayBandwidth configuration values:

- **RelayBandwidthRate:** This value represents the maximum average bandwidth per second that the relay will allow for network traffic. It should be set to a value lower than the average available network bandwidth to avoid congestion and to ensure stability (e.g. 75% of the average available bandwidth).

$$RelayBandwidthRate = 0.75 * 2874Mbps * 1.000.000\frac{bytes}{s} = 2.155.500.000\frac{bytes}{s}$$

- **RelayBandwidthBurst:** This value represents the maximum burst capacity the relay can handle. This occurs above the RelayBandwidthRate for short periods and should be set higher to accommodate short spikes in traffic (e.g. 20% higher than the RelayBandwidthRate).

$$RelayBandwidthBurst = 1.2 * RelayBandwidthRate = 2.586.600.000\frac{bytes}{s}$$

Finally, before running the *"tor"* command, adding a hashed password would provide additional security. The hashed output should be inserted in the *"HashedControlPassword"* segment from Snippet 5.30.

```
1 $ sudo tor --hash-password <your_password>
```

Snippet 5.31: Hash password

By using this command, one can generate a hashed password that can later be used in Tor's configuration file (*torrc*) to secure administrative access. The hashed password helps ensure that only users with the correct password can access certain functionalities or perform administrative tasks.

5.3.5 Tor Network Forensics

The first step in de-anonymizing a THS is to gain a clear picture of the traffic flow at every essential component. Therefore, traffic must be monitored at the attackers device and the malicious guard, respectively.

As a starting point, one must consider if there is a way to discern the difference between Tor and ordinary clear-net traffic.

Initial tests with Wireshark show no signs of Tor traffic other than the tell-tale signs of TCP and TLSv1.2 use, along with port 9001 and 9030. This is naturally due to the Tor protocols layered encryption approach. These signs would only appear occasionally, however. This is the result of what occurs at port 9030.

Port 9030 is utilized when the client is in contact with the directory server. If the attacker does not have a list of Tor relays stored locally, they will receive the list from the directory server. Therefore, the aforementioned tell-tale signs would only appear if the client was started for the first time, or after a long time.

Additionally, port 9001 is the default Tor port, although, many relays run on port 443 or others (This is configurable). Therefore, traffic will only be visible on this port, if the attacker contacts a relay with the default port settings.

Monitoring traffic on the attacking device did bare fruit, however. Although, network transfers that occur on the Tor network are encrypted, one can view un-encrypted network traffic destined for the Tor network. This refers to the traffic sent between localhost TCP sockets. Anonymous Tor browsing can be viewed, by loading a PCAP file with localhost traffic into NetworkMiner. This technique is referred to as TorPCAP[41].

TorPCAP

The crux of this technique relies on capturing Tor traffic before it gets Encrypted.

The Tor browser on the attacking machine includes a variety of installations. These installations include a SOCKS proxy listening on port 9150 on localhost. By utilizing the local SOCKS proxy, the Tor Browser can connect to the Tor network and have its traffic encrypted and forwarded. Therefore, by sniffing traffic on localhost, a forensic trail of dark-net traffic can be established.

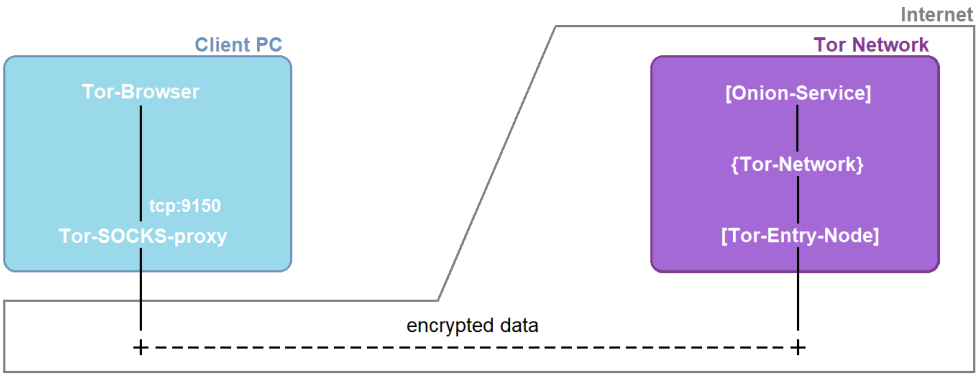

Figure 5.5: TorPcap[41]

In order to gain a clear view of the traffic an additional step must be taken. A tool to parse the SOCKS protocol is required. This is where NetworkMiner was introduced. NetworkMiner utilizes SOCKS parse features which can extract and reassemble data going to and from the Tor network.

The following images showcases some of the results of NetworkMiners contributions: Although, NetworkMiner has an abundance of features, Figure 5.6 depicts a user logging their credentials.

Anomalies

Hosts (24) | Files (24) | Images (2) | Messages | Credentials (6) | Sessions (40) | DNS (139) | Parameters (654)

☑ Show Cookies ☑ Show NTLM challenge-response ☐ Mask Passwords

Protocol	Username	Password
SOCKS	yfjcqmwksq5trzghb3ekiam6krszdb2wq7bmirmjxv...	13100161344062fea1ea60ae22251d179e
HTTP Cookie	connect.sid=s%3AdgnzeQeQBYtBZbGcl0oTg70...	N/A
MIME/MultiPart	test	test
HTTP Cookie	connect.sid=s%3A0Ny5O5r9O7gCyXuhbBX3rJr...	N/A
HTTP Cookie	connect.sid=s%3A0Ny5O5r9O7gCyXuhbBX3rJr...	N/A
SOCKS	--unknown--:0	7bb857aeffcd8939ff54f64e2401913a9ea

Figure 5.6: Credential Spotting

Additionally, Figure 5.7 depicts a variety of requests being issued to the THS.

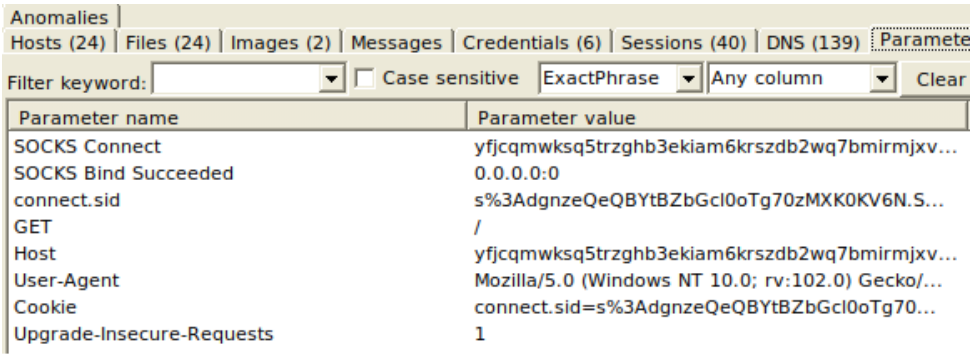

Figure 5.7: Parameter Names

The next step is to monitor traffic on the compromised relays.

Tool Selection

The following tools are to be considered when monitoring traffic on a Tor relay:

- **Nyx:** Nyx is a command-line tool for monitoring and managing Tor relays. It allows you to view real-time statistics and traffic of your Tor relay. You can install it by typing *"sudo apt-get install nyx"* in the terminal and then running it with the *"nyx"* command[42].
- **Arm:** Arm is another command-line tool that allows you to view and monitor traffic on your Tor relay. It provides extensive statistics and allows you to configure your relay settings. You can install it by typing *"sudo apt-get install arm"* in the terminal and then running it with the *"arm"* command[43].
- **Tor Metrics:** Tor Metrics is a web-based tool that provides detailed statistics about the Tor network, including the traffic and performance of individual relays. You can access it at [44].

The following evaluation matrix is meant to showcase the process in which the dark-net monitoring tool was selected. In this instance *"Availability"* refers to the labors that are required to acquire the tool, *"Attributes"* refers to the number of features the tool has available and *"Documentation"* refers to the amount of existing documentation.

	Evaluation		
Tools	**Availability**	**Attributes**	**Documentation**
Nyx	Medium	High	High
Arm	Medium	High	Low
Tor Metrics	High	Low	High

The different scores can be described by the following table:

Ratings			
Score	**Availability**	**Attributes**	**Doc**
High	No payments or set-up required	Monitoring, filtering & more	A lot
Medium	Free, yet some set-up required	Monitoring & filtering	Some
Low	Payment & set-up required	Monitoring	None

Ideally, the tool of choice should have a rating of *"High"* in all categories.

As the evaluation matrix suggests, *"Nyx"* shall be the monitoring tool of choice for the remainder of the paper.

However, after further research, it became apparent that the tools *"Nyx"* and *"arm"* are virtually identical. That being said, the bulk of online documentation refers to *"Nyx"* when these tools are mentioned.

Flags & *Nyx* Use-cases

By installing the *"Nyx"* dependency, Tor traffic passing through the relay can be viewed in different formats including the graphs seen in figure 5.8.

```
1 $ sudo apt install nyx
```

Snippet 5.32: Intallation

Figure 5.8 shows the guards usage represented in Nyx. Ultimately, a Tor relays use-cases are dictated by its acquired flags (as seen in figure 5.8). A Tor relay acquires different flags, such as ***"Fast"***, ***"Running"***, etc., based on its performance and characteristics. These flags are assigned by the Tor's directory authorities. By deploying a Tor relay via Linode, one can receive the following flags within a week: ***Fast, Running, Stable, HSDir, V2Dir, Valid & Guard.***

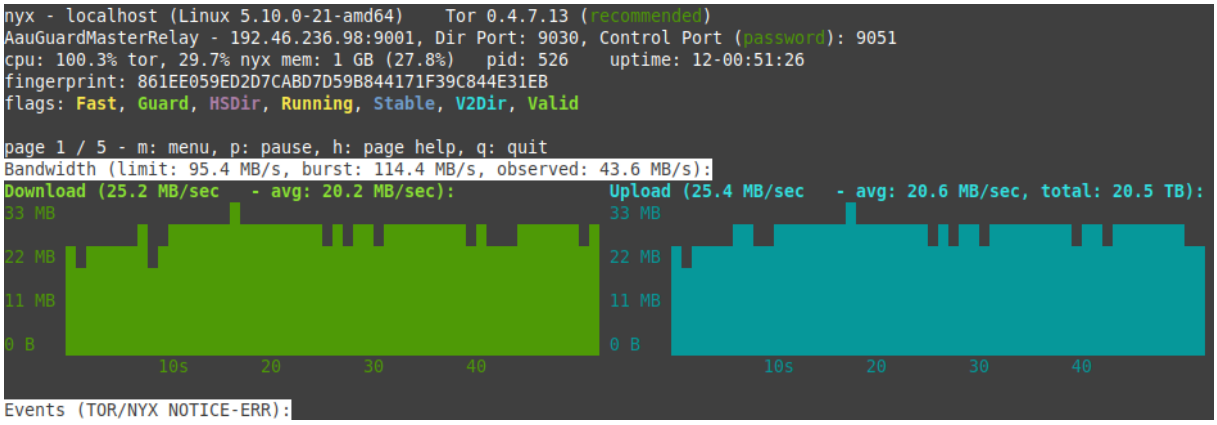

Figure 5.8: Relay Bandwidth Graph

It should be noted that the Tor relay was configured to accommodate a guard as opposed to an exit node. Therefore, the same rules do not necessarily apply for flags such as ***"Exit"***.

The following will briefly explain each flag received and how the Tor relay obtained them[45]:

- **Fast:** Tor relays are marked with this flag if they meet certain bandwidth requirements. Specifically, they need to have a high enough observed bandwidth to handle Tor network traffic effectively. The bandwidth measurements are collected by the directory authorities and updated periodically (a couple of hours).
- **Running:** This flag simply indicates that a relay is currently operational and accepting Tor connections. If the relay is online and functioning properly, it will receive this flag.
- **Stable:** To receive this flag, a relay needs to have been running continuously for a specified amount of time, (typically around four days). This flag helps ensure that the relay has demonstrated stability and reliability over an extended period.
- **Valid:** This flag is assigned to relays that meet certain criteria and pass basic checks. It indicates that the relay is considered valid and can be included in the Tor network's consensus document, which is a shared view of the network's relays maintained by the directory authorities. This flag is typically one of the first flags a relay receives.
- **V2Dir:** This flag is given to relays that are capable of serving as a directory cache for Tor clients. These relays maintain a copy of the Tor network's directory information and can respond to directory requests from clients. They play a role in distributing directory information and reducing the load on the directory authorities.
- **HSDir:** This flag is assigned to relays that are eligible to store and serve as introduction points for THSs. These relays store and provide information about THSs, allowing clients to connect to them.
- **Guard:** Finally, this flag is given to relays that are deemed suitable for use as an entry point (guard) into the Tor network. Guards are expected to have good uptime, bandwidth, and stability. They serve as the first hop in the Tor circuit and are crucial for user privacy and security.

Once the ***"Guard"*** flag in particular has been acquired, a stable connection to the THS can be made as displayed in figure 5.9

```
> tor@1.0.0 start
> nodemon server.js

May 18 13:23:11.000 [notice] Bootstrapped 10% (conn_done): Connected to a relay
May 18 13:23:12.000 [notice] Bootstrapped 14% (handshake): Handshaking with a relay
[nodemon] 2.0.19
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
May 18 13:23:12.000 [notice] Bootstrapped 15% (handshake_done): Handshake with a relay done
May 18 13:23:12.000 [notice] Bootstrapped 75% (enough_dirinfo): Loaded enough directory info to build circuits
May 18 13:23:12.000 [notice] Bootstrapped 80% (ap_conn): Connecting to a relay to build circuits
May 18 13:23:13.000 [notice] Bootstrapped 85% (ap_conn_done): Connected to a relay to build circuits
May 18 13:23:13.000 [notice] Bootstrapped 89% (ap_handshake): Finishing handshake with a relay to build circuits
May 18 13:23:14.000 [notice] Bootstrapped 90% (ap_handshake_done): Handshake finished with a relay to build circuits
May 18 13:23:14.000 [notice] Bootstrapped 95% (circuit_create): Establishing a Tor circuit
May 18 13:23:15.000 [notice] Bootstrapped 100% (done): Done
```

Figure 5.9: THS establishing a connection to Guard

Now that a connection has been established, more details can be viewed through Nyx when accessing the THS as a client. Nyx provides information about the relays involved in Tor connections, including their IP addresses. These details can be viewed by entering the Nyx *"connections"* option which shows all inbound and outbound network traffic for Tor.

Specifically, Nyx can show the IP addresses of the Tor relays that are part of the circuit the Tor connection is currently using. It allows users to monitor the performance and status of these relays, such as their bandwidth usage, uptime, and other relevant metrics.

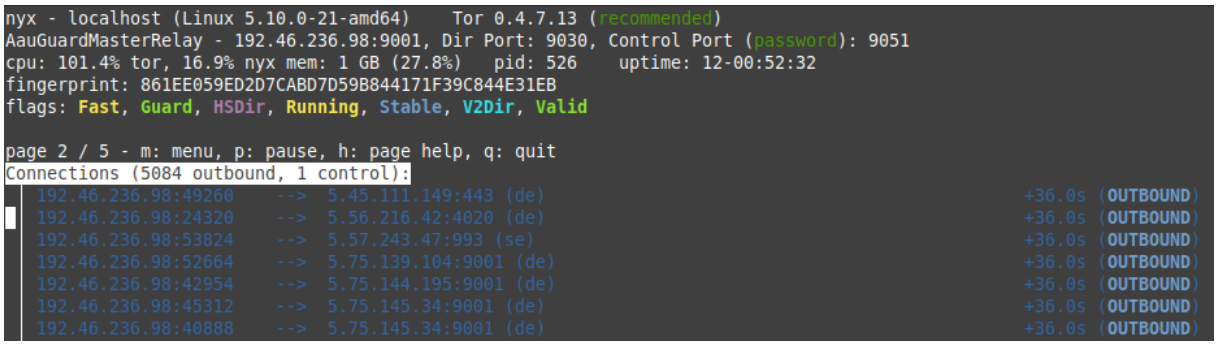

Figure 5.10: Relay Connections

Nyx primarily focuses on providing a comprehensive overview of the Tor network and its connections, rather than specifically analyzing traffic direction. That being said, it does have a *"sorting"* option. Figure 5.11 reveals the order in which traffic is viewed by default in addition to the Nyx tool's ability create new orders.

A category of sorting which is notably absent is a size option. A size option would be particularly beneficial when analyzing packet sizes for watermarks. This only applies for clear-web applications, however. As mentioned section 4.1, Tor cells are packets sent through the Tor network in an encrypted format with fixed sizes. Specifically, these encrypted messages are 512 bytes long. This ensures that the analysis of packet sizes becomes a non-factor.

Ordinarily, comprehensive forensics techniques such as Deep Packet Inspection (DPI) would help in identifying embedded watermarks. Unfortunately, as the cells are all encrypted and have fixed sizes, there is no discernible way to differentiate between Tor cells based on their features.

Therefore, Nyx's sorting feature does not appear to be particularly useful for the purposes of this experiment.

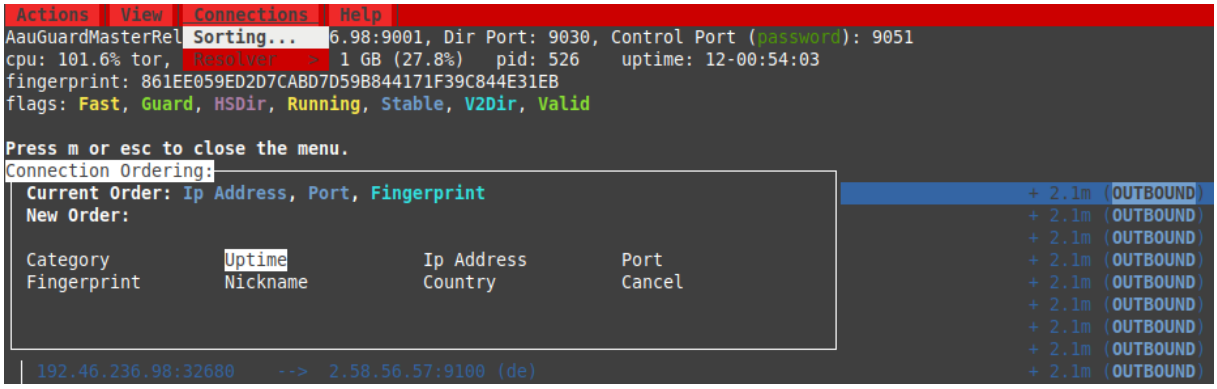

Figure 5.11: Traffic Sorting

5.3.6 Signal Detection

This section will describe the various methodologies in which signal embedding and detection can occur. Ideally, by embedding a signal at the client-side of a THS interaction, that same signal should be identified at the compromised guard.

Embedding

Figure 5.12 depicts an embedded watermark on a HTTP request. As the figure showcases, all that is required for a user to do this is a Tor browser. By accessing the THS in the browser and utilizing the ***Inspect*** feature (**right clicking** in the browser) one can modify HTTP requests in the browser.

In this case the payload *hello world* was added to an ordinary HTTP request of the THS's captcha page. The phrase *"hello world"* would be the watermark in this case e.g. a unique identifying piece of information pertaining exclusively to the packet in question.

By viewing the *Network* section of the inspector and specifying the aforementioned payload, the newly modified HTTP request is sent.

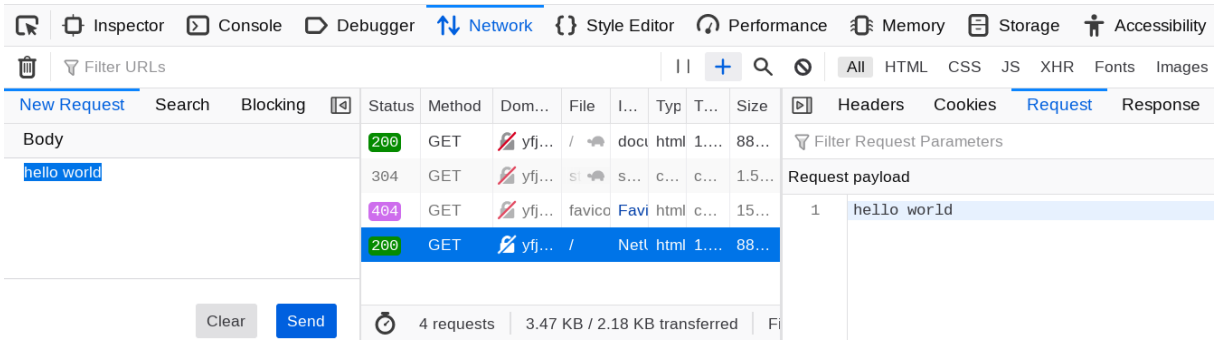

Figure 5.12: Embedding Payload

Now that a watermark has been added to the request, a method of detection must be devised. Ordinarily, performing DPI on the Tor relays incoming traffic would allow the attacker to derive the THS's IP address.

Matching

DPI involves examining the full packet payload, including the header and data sections, to extract information and perform various actions based on the content.

Additionally, DPI goes beyond simple packet header analysis (e.g. source and destination IP addresses, port numbers, etc.) and delves into the actual payload of the packets. By examining the packet contents, DPI can inspect and analyze various protocols, applications, and even individual payload data.

By utilizing the aforementioned TorPCAP method of forensics, the modified packet can be spotted on the client-side. The first step is to specify the Wireshark filter:

```
1 data contains "hello world"
```

Snippet 5.33: Wireshark Filter

Viewing only *localhost* activity, the watermark is immediately detected as displayed in figure 5.13:

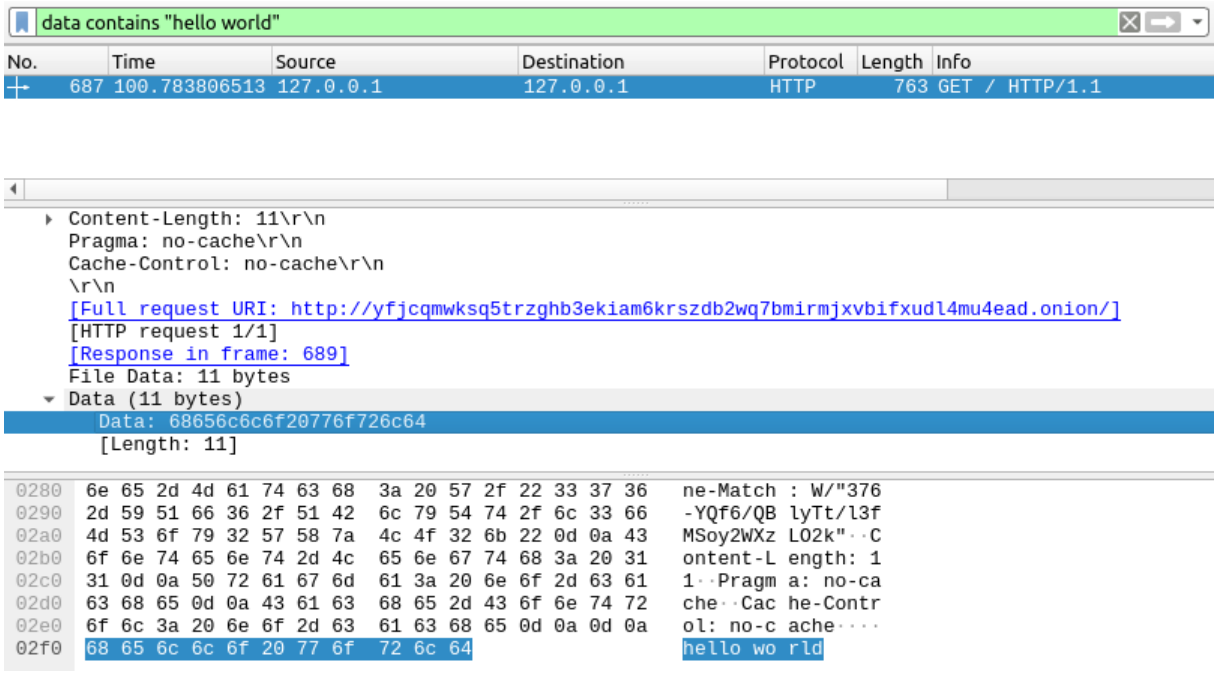

Figure 5.13: Monitoring Payload

This method of DPI cannot be used on the compromised Tor relays traffic, however. As described in figure 5.5, viewing *localhost* data transfers only applies on the client-side. However, if viewing packets in a similar manner on the THS's guard were feasible, one could determine the THS's IP address simply by viewing the destination IP address of the packet.

5.4 Evaluation

This section aims to discuss the takeaways from the PoC and evaluate the different factors which contributed to the projects outcome.

The objective of this PoC was to leak a THS's IP address by exploring the possibility of detecting watermarks via its compromised guard.

The assessments gathered revealed that embedded watermarks could not be detected by darknet monitors on the malicious guard. This was due to the encrypted Tor cells and their uniform sizes. The fact that every cell's length was identical made it challenging for the guard to discern any differences. As a result, no IP addresses were leaked from the THS, indicating a setback to achieve the intended goal. However, it is important to note that this PoC assumes the Tor network operates correctly and does not consider potential vulnerabilities or attacks on its encryption.

Nonetheless, this evaluation emphasizes the importance of the lessons learned from the project.

Despite the efforts made to detect watermarks on the THS's guard, the indiscernible nature of Tor cells proved to be effective in maintaining the anonymity of the THS. While this outcome may be disappointing in terms of achieving the project's objective, it is essential to recognize the value of these findings and to highlight the robustness of the Tor network's design and its ability to protect user privacy.

Additionally, an emphasis must be placed on the limitations of existing research. The theoretical nature and lack of detail regarding experimental procedures (tools used, successful screenshots etc.) result in an inability to reproduce the attack methods.

These insights highlight the need for additional experimentation and documentation in this area. The hope of this project is to enable researchers to develop more comprehensive documentation and practical approaches for de-anonymization of THSs.

Discussion 6

This chapter will reflect on the findings from previous chapters and convey the difficulties faced.

6.1 Reflections

This section touches on the difficulties involved in conducting THS de-anonymization research.

Initially when conducting the experiment, it became apparent that its most time consuming aspects pertained to the deployment of the malicious guard.

Troubleshooting the many errors found could be describe as nothing less than a *"wild goose chase"*. This was due to the abstract and vague descriptions of a Tor relay's life cycle[46]. A relay's life cycle refers to the amount of expected waiting time during the various stages of deployment. Particularly, it was difficult to gauge how long it would take for a relay to be listed in the Tor Metrics website after deployment. When listed in Tor Metrics, one can be sure the relay has been up and running. Therefore, it was difficult to know whether or not the relay was connected to the Tor network and operating correctly. It should be noted for future researchers, a relay should be visible in the Tor Metrics after a matter of hours (1-3 hours). Additionally, the waiting period to receive the required guard flag is approximately 1 week (assuming it was configured correctly).

Extensive time was spent on deploying the relay as it initially was deployed on Replit with the following directory structure:

```
./
├── .torrc
├── .replit
├── README.md
├── replit.nix
└── run.sh
```

However, that idea was halted indefinitely as Replit restarts its services every time one exits the running relay.

The Raspberry pi initially appeared to be the best option as it is design to run for long periods of time while not requiring monthly subscription payments. Unfortunately, network firewalls prohibited any Tor based experimentation.

Finally, after extensive testing on faulty systems, a cloud provider proved to be the option of choice.

In addition to the aforementioned time delays, a lack of communication from the Tor research

community was a clear detriment. Including the aforementioned comments left in the threads of Tor developer forums, the following e-mail was sent out to the authors of papers researching signal embedding or watermarking:

> *"Hi XXXX,*
>
> *My name is Clifford and I'm a Masters student at Aalborg University campus Copenhagen (Denmark). I'm currently writing my thesis on de-anonymizing hidden services.*
>
> *I've read your paper on "XXXXXXXXXXXX" and had some follow up questions. I was wondering if you were available for an online teams meeting. Any time would suit me.*
>
> *Best regards Clifford-Nelly Ndayikengurukiye"*

Unfortunately, there were no replies to any of the e-mails. The need to address these authors was a necessity as no paper displayed a comprehensive PoC. When considering the papers, each of them shared a common issue wherein, they all lacked sufficient detail in describing their experimental methodology. This rendered them overly abstract and non-reproducible. The absence of crucial information hampers the ability of other researchers to replicate the experiment and validate its findings.

6.2 Future Work

This section will suggest potential avenues future researchers could consider when de-anonymizing THSs.

What ultimately decides whether a relay can be trusted or not are the Tor directory authorities.

Directory authorities are trusted relays that maintain a list of other currently-running relays[47] They periodically publish a consensus alongside other directory authorities. Their primary responsibility is to create and maintain the network's consensus.

The consensus is a document that lists all of the currently active Tor relays, along with their various attributes and characteristics[48]. The consensus is updated regularly and is used by Tor clients to select relays for their circuits.

To create the consensus, directory authorities *"vote"* independently on which relays should be included in the consensus and how much influence each relay should have. The directory authorities then compare their votes and create a final consensus that reflects the collective judgment of the network.

Whether or not a relay can join the consensus is determined by its *"consensus weight"*. Consensus weight is a measure of how much trust a particular Tor relay has in the network's consensus. Consensus weights are assigned to each relay by the directory authorities.

Relays with higher consensus weights are considered to be more trustworthy and are more likely to be added to the consensus and used by Tor clients to create circuits throughout the network.

A major success of this project was configuring the malicious guard effectively so one could receive the highest possible consensus weight in the shortest amount of time. Therefore, takeaways for future work may include **ensuring** the THS connects to the malicious guard on its own accord.

Naturally, a relay with a high consensus weight has a higher chance of being selected, however, in a realistic setting, playing the odds is not a viable option.

Connecting itself would be ideal, however, there is currently no way to ensure a THS utilizes a specific guard. If this project were to continue, injection attacks would be the most appropriate option to accomplish this.

Another option would be to exploit a THS for application specific vulnerabilities. By doing this, an attacker could gain access to the THS via a shell and discern the IP address from there. This method is not optimal as it would vary from target to target as applications do.

In conclusion, when it comes to de-anonymizing THSs, it is evident that there are numerous approaches and techniques available. The question then becomes, can an attack method be executed practically?

Conclusion 7

This chapter will conclude the thesis and summarize its findings.

The lessons learned from this project are invaluable and can contribute to the broader understanding of anonymity and security within Tor.

The projects initial goal was to de-anonymize a THS by exploring the detection of watermarks from a compromised guard relay.

By ensuring the target utilized the same compromised guard for every constructed circuit, one could have a direct line of contact to the THS. Therefore, when embedding watermarks in client-side HTTP requests, the altered packet destined for the THS would be received by the compromised guard before relaying it. Theoretically, the THS's IP address could be leaked by an attacker reviewing the headers of the watermarked packets if they had a direct link to the THS. This must be done from the guard, however.

The PoC revealed that watermarks could not be detected by relays due to the encrypted Tor cells being of a uniform size. This finding highlighted the robustness of the Tor network's design and its ability to protect user privacy. While this result may not align with the project's initial goal, it emphasizes the effectiveness of the existing security measures within the Tor network.

In conclusion, the project's setbacks serve as a reminder that practical challenges often differ from theoretical expectations. It highlights the importance of conducting thorough and replicable experiments to bridge the gap between theory and practice. In addition, it is apparent there is a lack of detailed and reproducible documentation within the field. By sharing the lessons learned and the limitations encountered during this project, researchers can contribute to the collective knowledge and aid further investigations into de-anonymization techniques.

On a final, it is crucial to acknowledge that de-anonymization must always be conducted within the boundaries of legal and ethical frameworks. Privacy rights, freedom of expression, and the need for responsible disclosure must be respected and upheld. It is imperative for law enforcement agencies and security researchers to find a balance between uncovering illicit activities and ensuring the protection of individual rights and civil liberties.

Bibliography

[1] Open Source. *History*. URL: https://www.torproject.org/about/history/.

[2] Dr Mike Pound. *TOR Hidden Services - Computerphile*. 2017-06. URL: https://www.youtube.com/watch?v=lVcbq_a5N9I&t=548s (visited on 2017-06-09).

[3] Open Source. *Some statistics about onions*. 2015. URL: https://blog.torproject.org/some-statistics-about-onions/.

[4] Eugene Kaspersky. *What is the Deep and Dark Web?* URL: https://www.kaspersky.com/resource-center/threats/deep-web.

[5] Google. *Google Scholar*. URL: https://www.scholar.google.com (visited on 2022-10-17).

[6] Albert Kwon. *Circuit Fingerprinting Attacks: Passive Deanonymization of Tor Hidden Services*. 2015. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/kwon (visited on 2022-11-05).

[7] Andriy Panchenko. *Analysis of Fingerprinting Techniques for Tor Hidden Services*. 2017. URL: https://dl.acm.org/doi/abs/10.1145/3139550.3139564?casa_token=uanxg_Z3lbUAAAAA:P8uRGiIGt_Zp2460lXqKyCQpgiWmmb5N6adBCrY_Ml5-aHNowvuaxQHHROuj6NxNN1SeOFvA9WuR (visited on 2022-11-05).

[8] Asya Mitseva. *POSTER: Fingerprinting Tor Hidden Services*. 2016. URL: https://dl.acm.org/doi/abs/10.1145/2976749.2989054?casa_token=MsnBItE4GpkAAAAA:diavFdfoRsfGsRwpPnF3ObnWlPobJxB0nu9b7ECEWGLLw2ncXXLMmqsf60RGNokDk1wmAcXyn0a-Eg (visited on 2022-11-12).

[9] Marc Juarez Miro. *Fingerprinting Hidden Service Circuits from a Tor Middle Relay*. 2017. URL: https://www.research.ed.ac.uk/en/publications/fingerprinting-hidden-service-circuits-from-a-tor-middle-relay (visited on 2022-11-12).

[10] Muqian Chen. *Napping Guard: Deanonymizing Tor Hidden Service in a Stealthy Way*. 2020. URL: https://ieeexplore.ieee.org/abstract/document/9343014?casa_token=7-cvHB4VK6AAAAAA:jVNbSfVGEqcC6F_S12BsyIbjf5c0nG7g6vjBYSJJUkJZIoOT18Zwuz_fNorVXYDEBfLEEZA9 (visited on 2022-11-05).

[11] Muqian Chen. *SignalCookie: Discovering Guard Relays of Hidden Services in Parallel*. 2019. URL: https://ieeexplore.ieee.org/abstract/document/8969639?casa_token=8q6afyFpDXUAAAAA:fwl_RGZKR0YVG-U4dNBmxTdSVQCsoAaA6GxFF1gu7i9rFR0jHM3qHvnGBbhFsapFP_FjpzAn (visited on 2022-11-05).

[12] Alex Biryukov. *Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization*. 2013. URL: https://ieeexplore.ieee.org/abstract/document/6547103?casa_token=9SMp1Sloz54AAAAA:i522FNoVpOqHTxdA1S4SnLgCV6uytsL2b0Td2RZv5LbnBu0QZwWyslHAwaBm6L9m (visited on 2022-11-05).

[13] Sabita Nepal. *Deanonymizing schemes of hidden services in tor network: A survey*. 2015. URL: https://ieeexplore.ieee.org/abstract/document/7057949?casa_token=WjWdwIVr41sAAAAA:DQ5Cg5TTpbKxTprQSRJTrxKiNcqA8XQO8yzVJEmplioJ0rYaK-bJb4Q_J8_RiK0opsWAiw20 (visited on 2022-11-05).

[14] Diana L. Huete Trujillo. *Tor Hidden Services: A Systematic Literature Review*. 2021. URL: https://www.mdpi.com/2624-800X/1/3/25 (visited on 2022-11-12).

[15] Srdjan Matic. *CARONTE: Detecting Location Leaks for Deanonymizing Tor Hidden Services*. 2015. URL: https://dl.acm.org/doi/abs/10.1145/2810103.2813667?casa_token=41krh19r4g0AAAAA:1zppUIMwjbgaw2ICxaVVM-nMk3UbXWAjtpPmAgK2k5Ygy54o4mAfkxZxs_KqsaRoF4jGiCDOoKFt (visited on 2022-11-05).

[16] Rob Jansen. *The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network*. 2014. URL: https://apps.dtic.mil/sti/citations/ADA599695 (visited on 2022-11-05).

[17] Amirali Sanatinia. *Off-path man-in-the-middle attack on tor hidden services*. 2017. URL: https://www.ccs.neu.edu/home/amirali/publications/tor_mitm_nesd.pdf (visited on 2022-11-05).

[18] Alfonso Iacovazzi. *Inflow: Inverse Network Flow Watermarking for Detecting Hidden Servers*. 2018. URL: https://ieeexplore.ieee.org/abstract/document/8486375?casa_token=Akr41j9m3jUAAAAA:w8Js61nmxpVLXSPPyxMV0Z8ml0K-nU_3MoatfsC-LSCwdcJ2EJDEa6BbD9JAfc_8DObresbadQ (visited on 2022-11-05).

[19] Varun Nair. *Inventive Computation and Information Technologies*. 2022. URL: https://link.springer.com/chapter/10.1007/978-981-16-6723-7_3 (visited on 2022-11-12).

[20] Wenlin Han. *Darknet and Bitcoin De-anonymization: Emerging Development*. 2020. URL: https://ieeexplore.ieee.org/abstract/document/9161431 (visited on 2022-11-12).

[21] Marco Simioni. *Investigative Techniques for the De-Anonymization of Hidden Services*. 2021. URL: https://ieeexplore.ieee.org/abstract/document/9382354?casa_token=EnGJ6EP-SggAAAAA:QAThU6106ViopH3Pnw87ejnuuUnrxXuvS1nDuT30wXDQPfZijp4Uh6Vx7VShX-I-kpl-aMt_Cw (visited on 2022-11-12).

[22] Massimo Bernaschi. *Exploring and Analyzing the Tor Hidden Services Graph*. 2017. URL: https://dl.acm.org/doi/abs/10.1145/3008662?casa_token=aNgijYZTOWYAAAAA:Nkgkk1TnwII1OAXOj8x2huA0wcuk7NH3c6HJeHY_rbeweDWJjKA5sU2uZHHPcTdfIX7IdHUFOoOiSA (visited on 2022-11-12).

[23] Yi Qin. *Tracing Tor Hidden Service Through Protocol Characteristics*. 2022. URL: https://ieeexplore.ieee.org/abstract/document/9868859?casa_token=TXeIX6L6v10AAAAA:CCyUzHYqsfrhN-i7bPQkfP4mWqsOq0qqAgrO6hOvALnxYXq-MZUiHetMOuUOuGZuLsZLsewMGQ (visited on 2022-11-12).

[24] Johannes Ödén. *Deanonymizing Onion Services by Introducing Packet Delay*. 2022. URL: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1664834&dswid=6533 (visited on 2022-11-12).

[25] Jianjun Lin. *Deanonymizing Tor in a Stealthy Way*. 2019. URL: https://ieeexplore.ieee.org/abstract/document/8958750?casa_token=tU7Tn8dzrKgAAAAA:BU3encUg_fILDYIOTzI8hik8q8nW98jUbCXEWTo79Akk1G68b7NmQdLbZ7coeGHb_khrOq2caw (visited on 2022-11-12).

[26] Autumn Skerritt. *How Does Tor Really Work? The Definitive Visual Guide*. URL: https://skerritt.blog/how-does-tor-really-work/.

[27] Dr Mike Pound. *How TOR Works - Computerphile*. 2017-05. URL: https://www.youtube.com/watch?v=QRYzre4bf7I&t=1s (visited on 2017-05-31).

[28] Ashwin S. *How to Stay Anonymous on the Internet using TOR Network?* URL: https://www.hacker9.com/can-hide-online-using-tor-network/.

[29] Dr Mike Pound. *TLS Handshake Explained - Computerphile*. 2020-11. URL: https://www.youtube.com/watch?v=86cQJ0MMses (visited on 2020-11-06).

[30] Open Source. *How do Onion Services work?* URL: https://community.torproject.org/onion-services/overview/.

[31] Clifford-Nelly Ndayikengurukiye. *Pentesting a Vulnerable Hidden Service*. Academic Paper. 2022.

[32] "Tien Nguyen". *Node.js Express Login example with MySQL database*. URL: https://dev.to/tienbku/node-js-express-login-example-with-mysql-database-2n51.

[33] Arnav Kumar. *Deploy Your Node.js (or any) Apps to Dark Web! It's so Easy!* URL: https://dev.to/arnavkr/deploy-your-nodejs-or-any-apps-to-dark-web-its-so-easy-26el.

[34] "". *Replit*. URL: https://en.wikipedia.org/wiki/Replit.

[35] Roger Dingledine. *Client's choice of rend point can leak info about hidden service's guard relay*. URL: https://gitlab.torproject.org/tpo/core/tor/-/issues/14917.

[36] "". *iptables*. URL: https://en.wikipedia.org/wiki/Iptables.

[37] "". *about*. URL: https://www.linode.com/company/about/.

[38] "". *My relay or bridge is overloaded what does this mean?* URL: https://support.torproject.org/relay-operators/relay-bridge-overloaded/#metricsport.

[39] "". *Top Relays by Consensus Weight*. URL: https://metrics.torproject.org/rs.html#toprelays.

[40] "". *Linode 4GB by Linode*. URL: https://pcr.cloud-mercato.com/providers/linode/flavors/g6-standard-2/performance/network-bandwidth.

[41] Leon Kowalski. *TorPCAP - Tor Network Forensics*. URL: https://www.netresec.com/?page=Blog&month=2018-12&post=TorPCAP---Tor-Network-Forensics.

[42] "". *Welcome to Nyx*. URL: https://nyx.torproject.org/.

[43] "". *Man Pages*. URL: https://manpages.ubuntu.com/manpages/bionic/man1/arm.1.html.

[44] "". *Analysis*. URL: https://metrics.torproject.org/.

[45] "". *Tor directory protocol, version 3*. URL: https://github.com/torproject/torspec/blob/main/dir-spec.txt.

[46] "arma". *The lifecycle of a new relay*. URL: https://blog.torproject.org/lifecycle-of-a-new-relay/.

[47] "". *Directory Authority*. URL: https://support.torproject.org/glossary/directory-authority/.

[48] "". *consensus*. URL: https://support.torproject.org/glossary/consensus/.

Relay Machine Info A

AauGuardMasterRelay - Machine Statistics

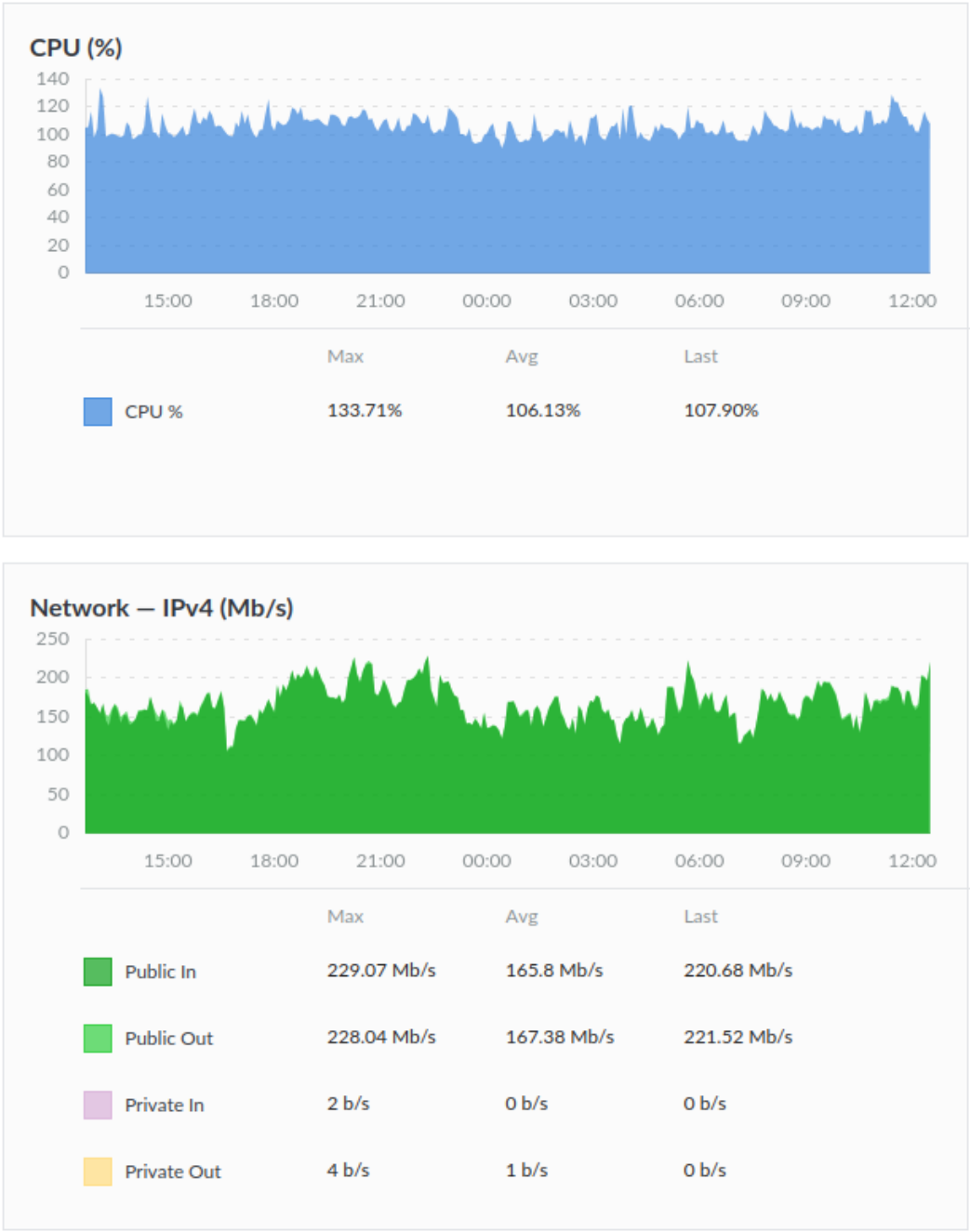

Figure A.1: Relay VM Analytics (1)

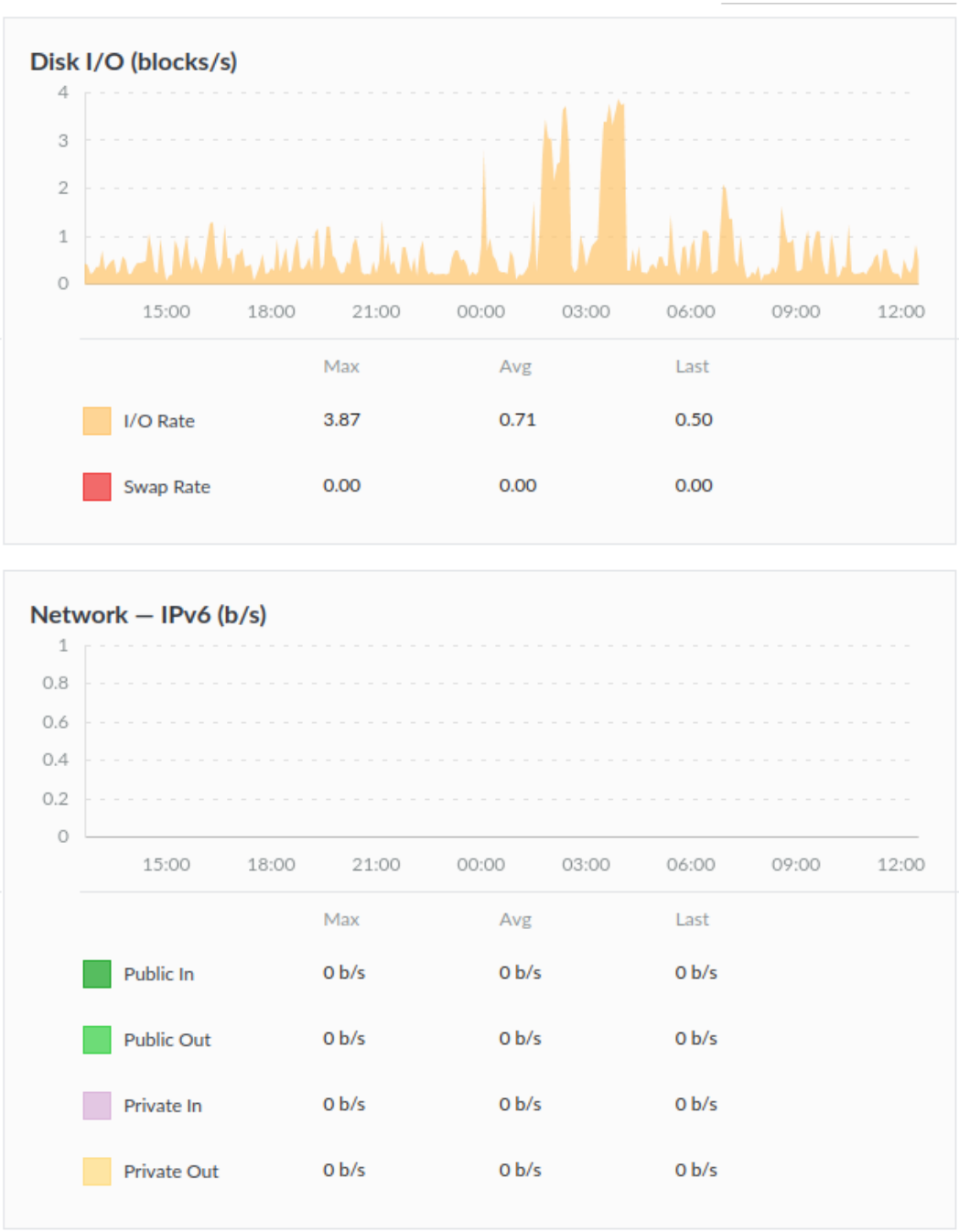

Figure A.2: Relay VM Analytics (2)

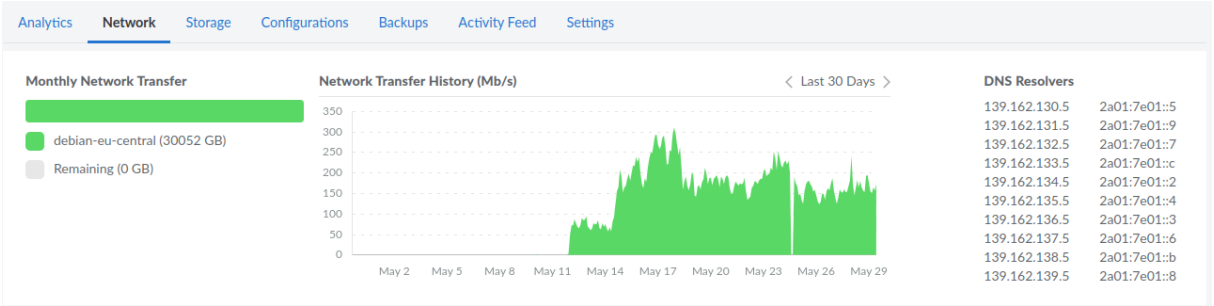

Figure A.3: Relay VM Network Traffic

MasterRelayAauGuard - Machine Statistics

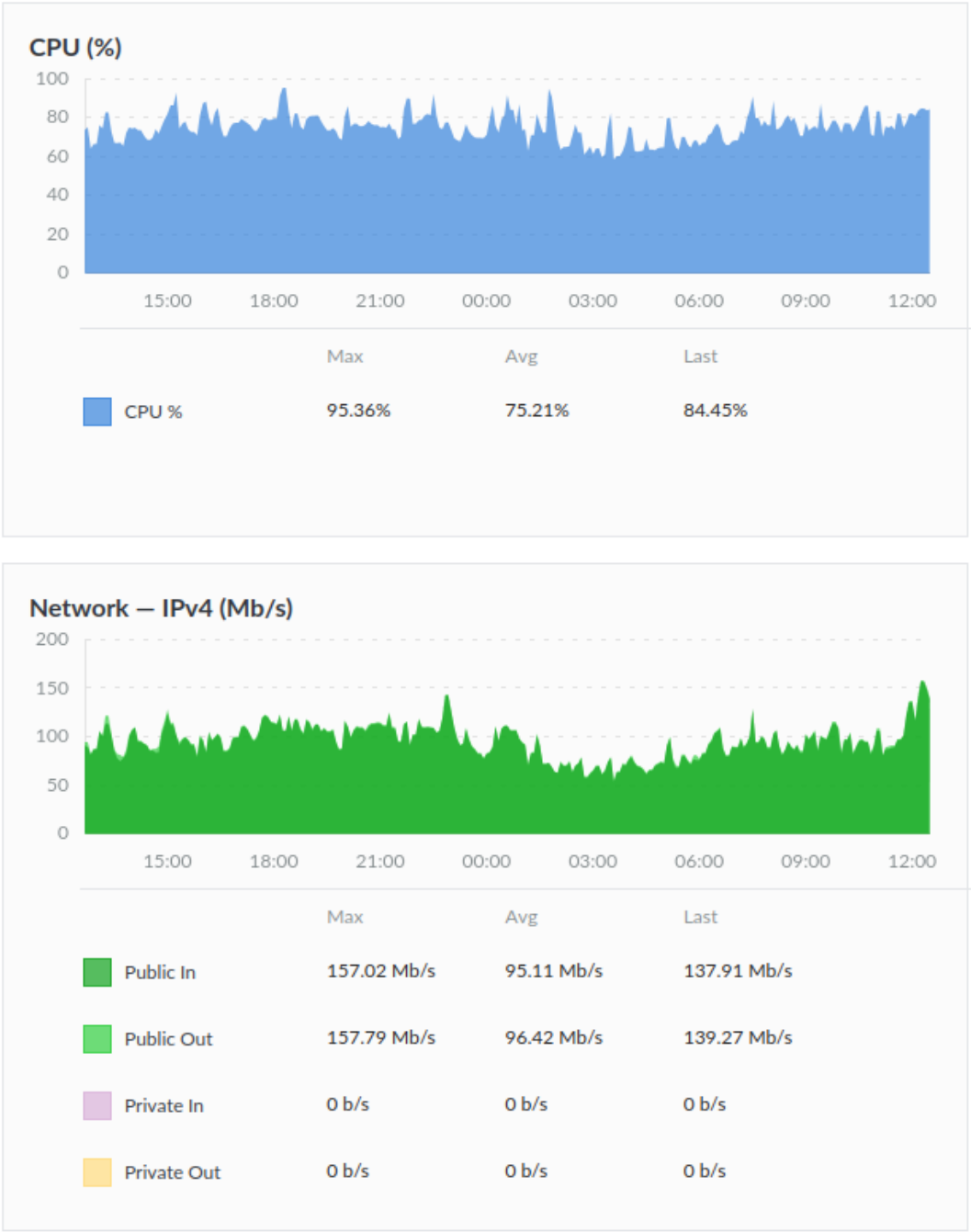

Figure A.4: Relay VM Analytics (1)

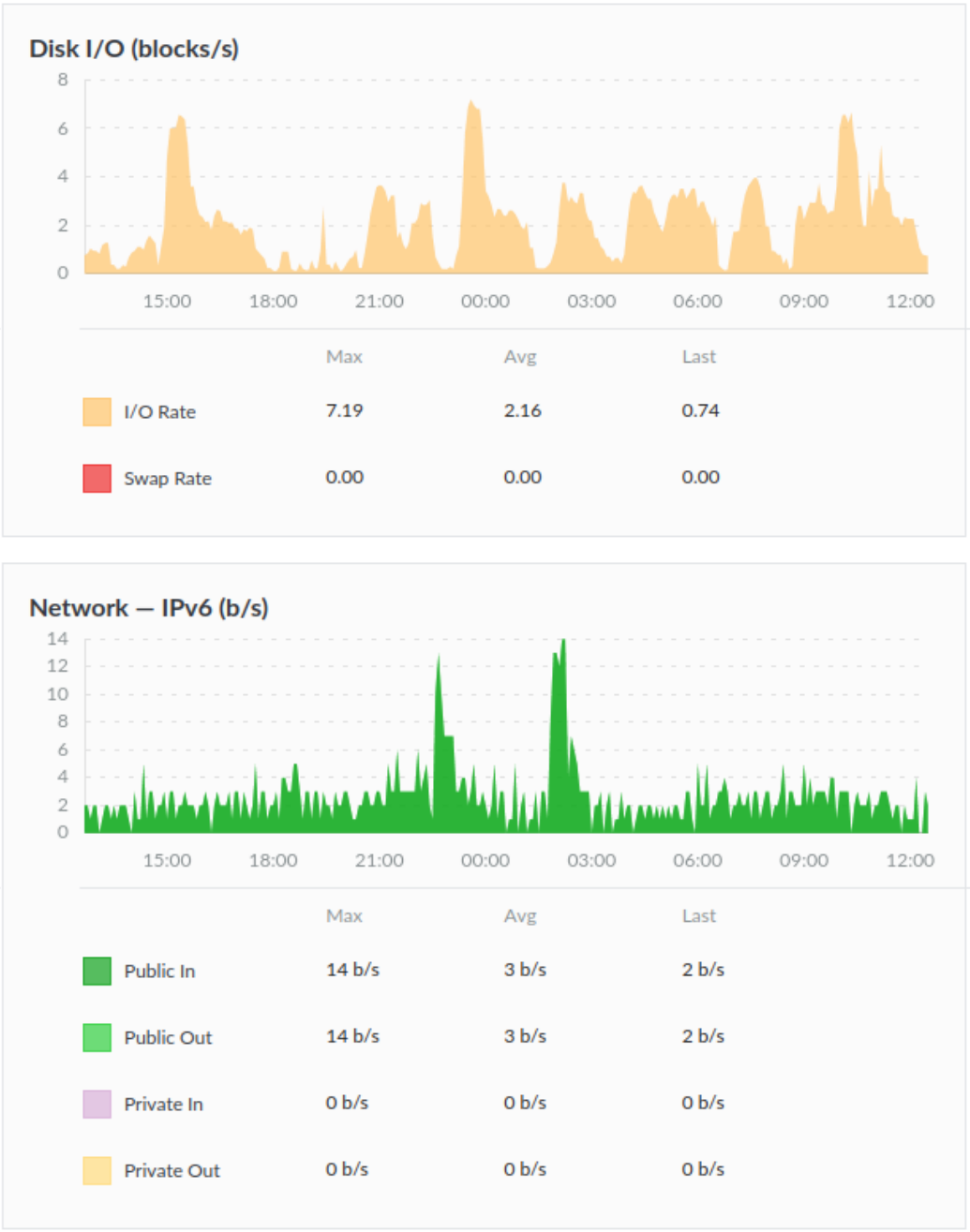

Figure A.5: Relay VM Analytics (2)

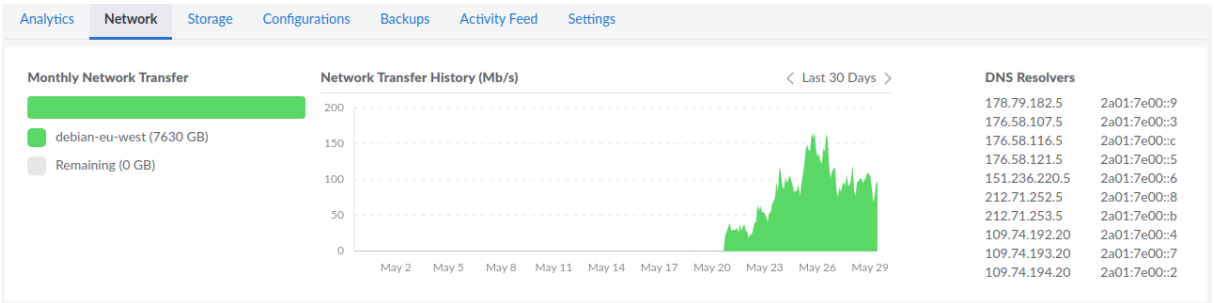

Figure A.6: Relay VM Network Traffic

Relay Metrics B

AauGuardMasterRelay - Specifications

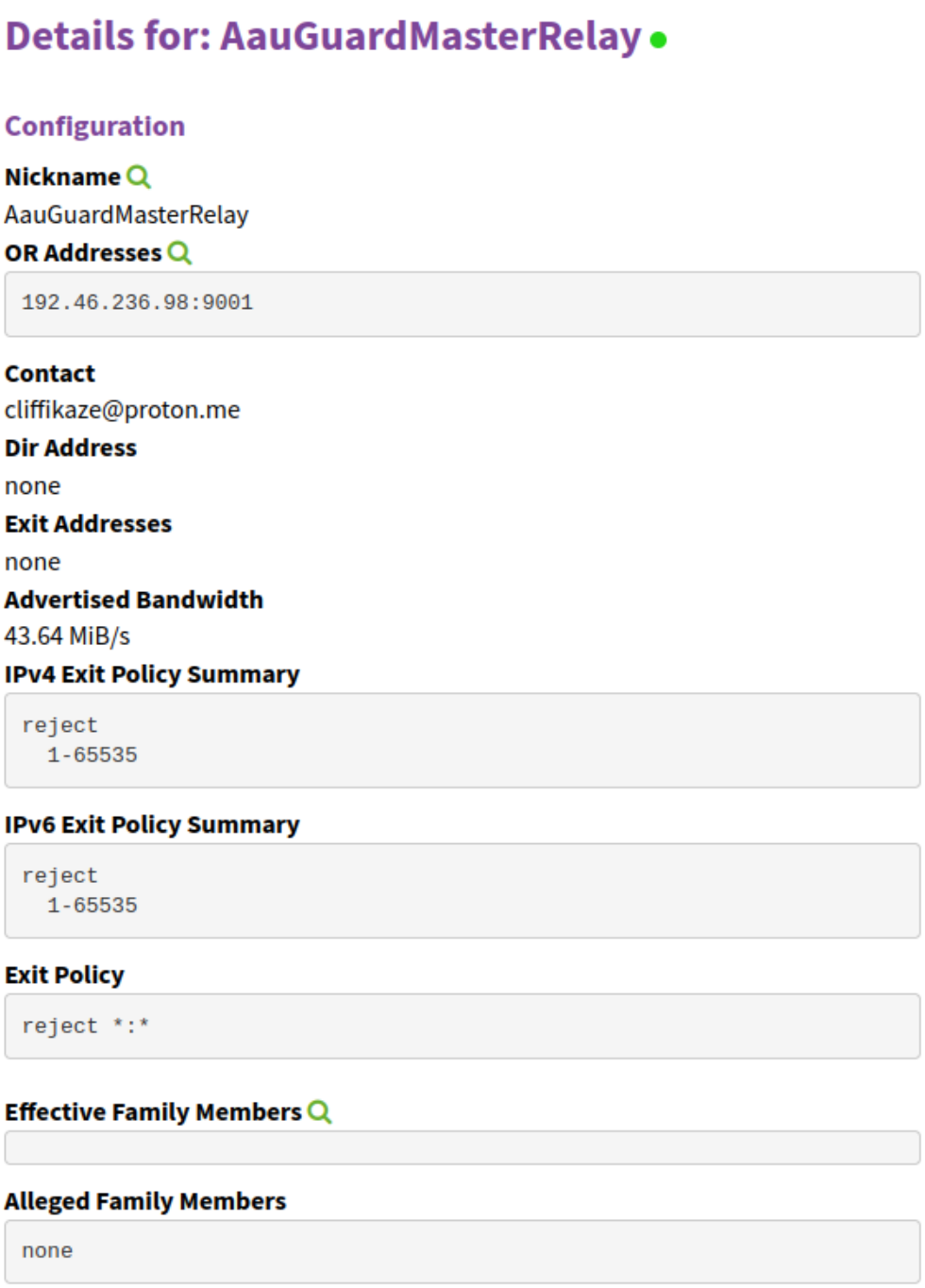

Figure B.1: Relay Configurations[39]

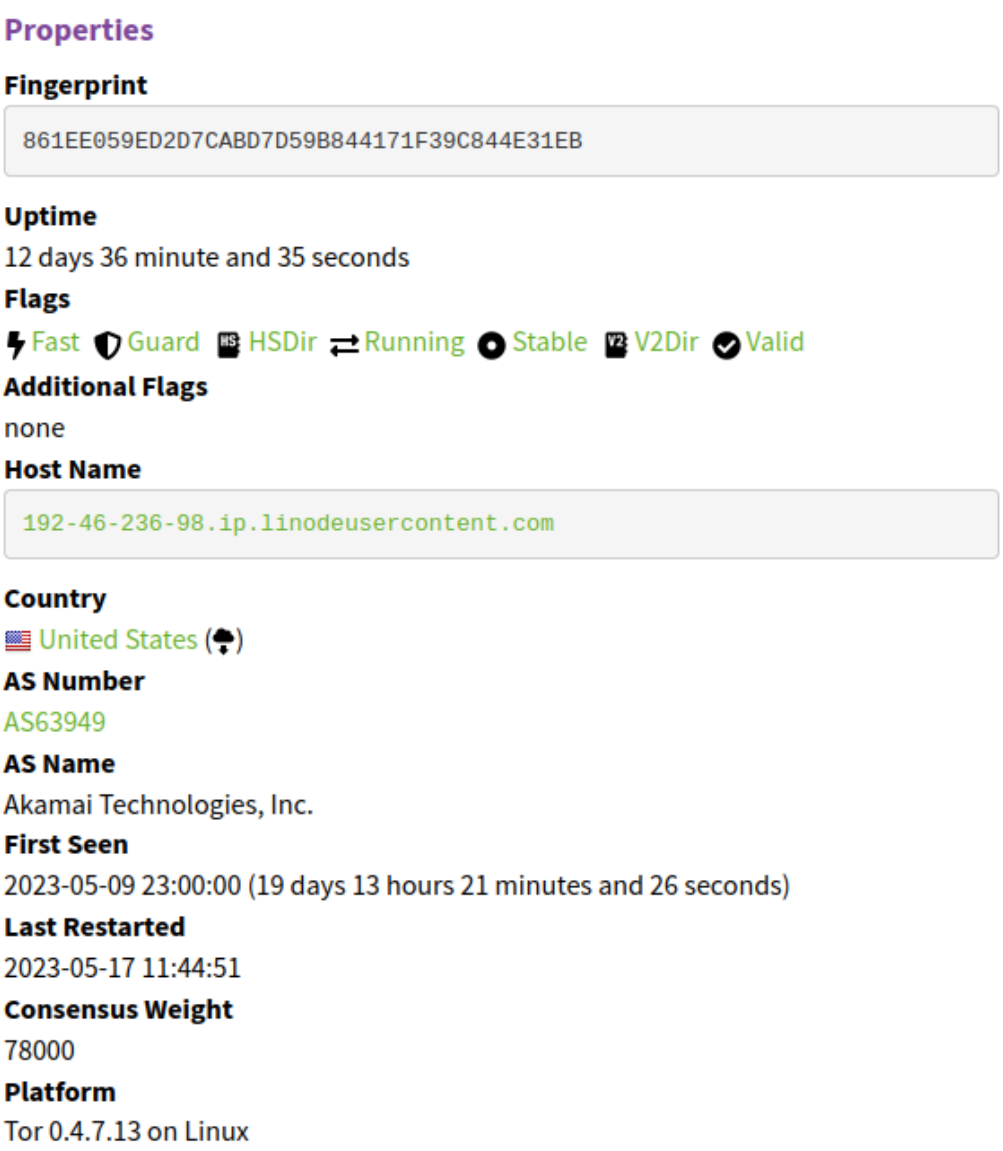

Figure B.2: Relay Properties[39]

MasterRelayAauGuard - Specifications

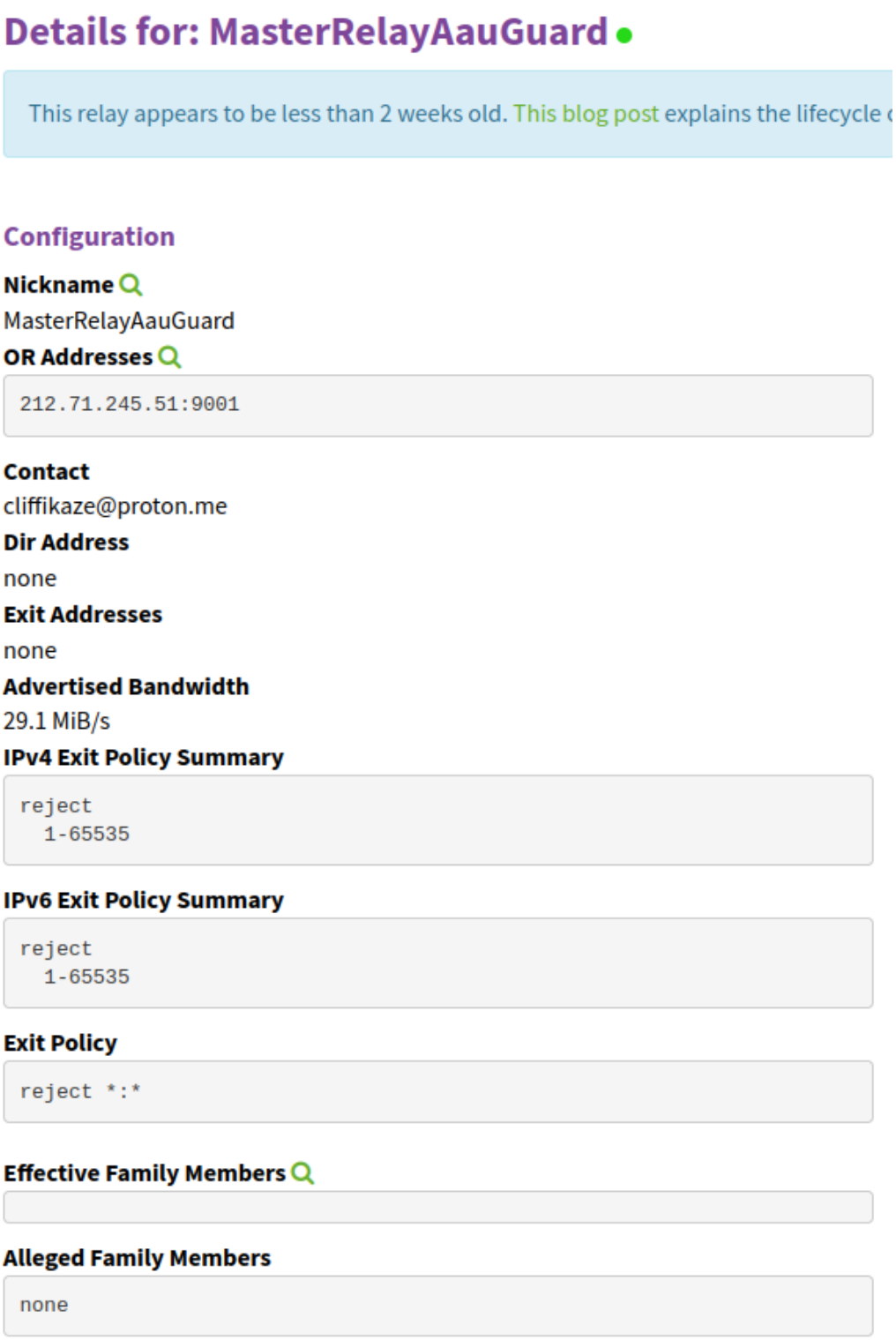

Figure B.3: Relay Configurations[39]

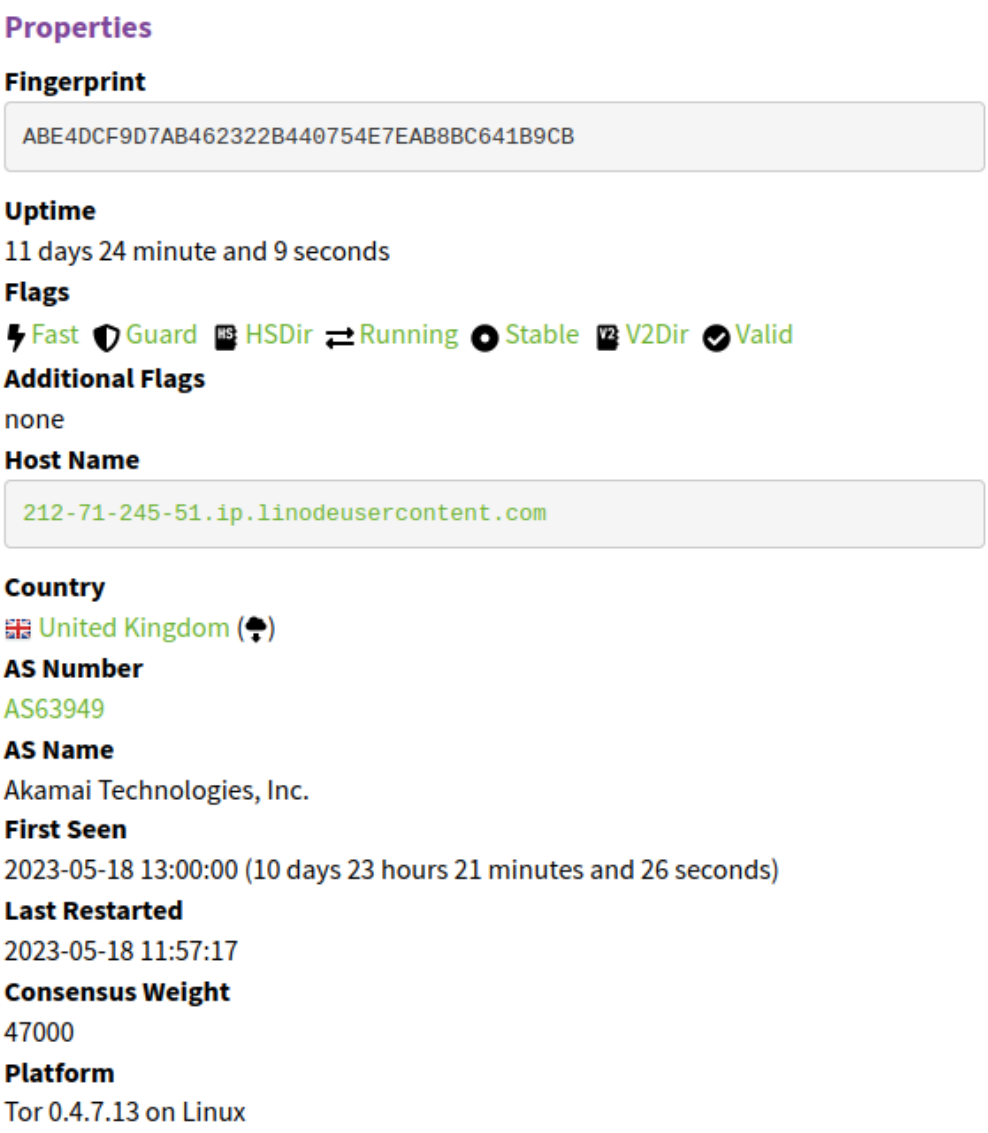

Figure B.4: Relay Properties[39]

AauGuardMasterRelay - Graphs

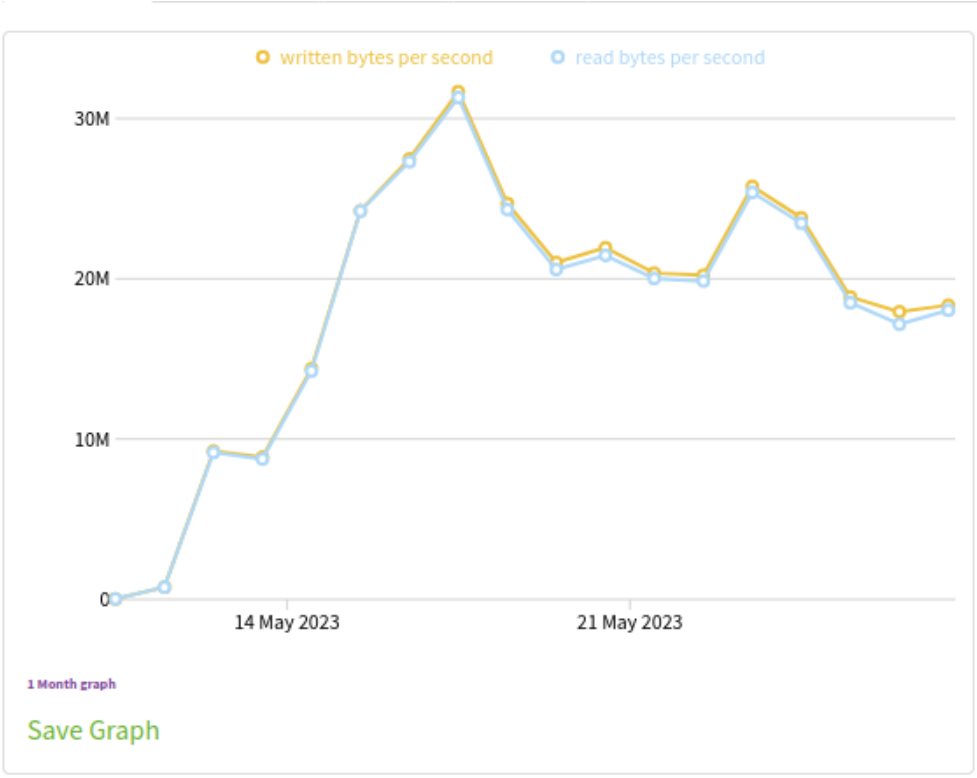

Figure B.5: Data Usage Evolution[39]

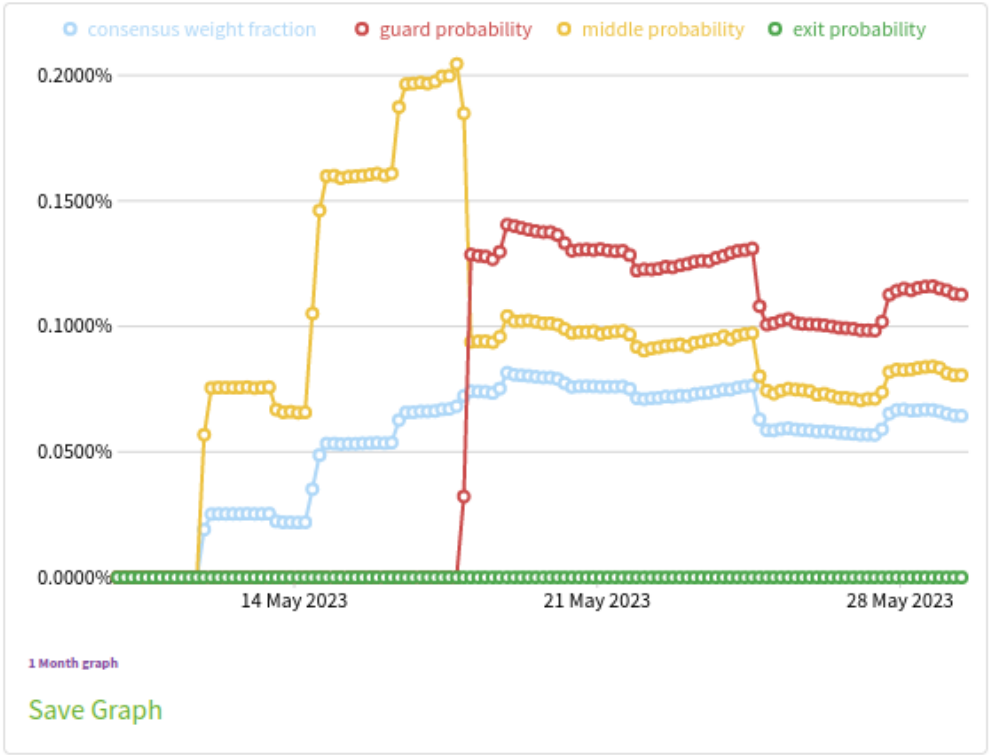

Figure B.6: Flag Assignment Evolution[39]

MasterRelayAauGuard - Graphs

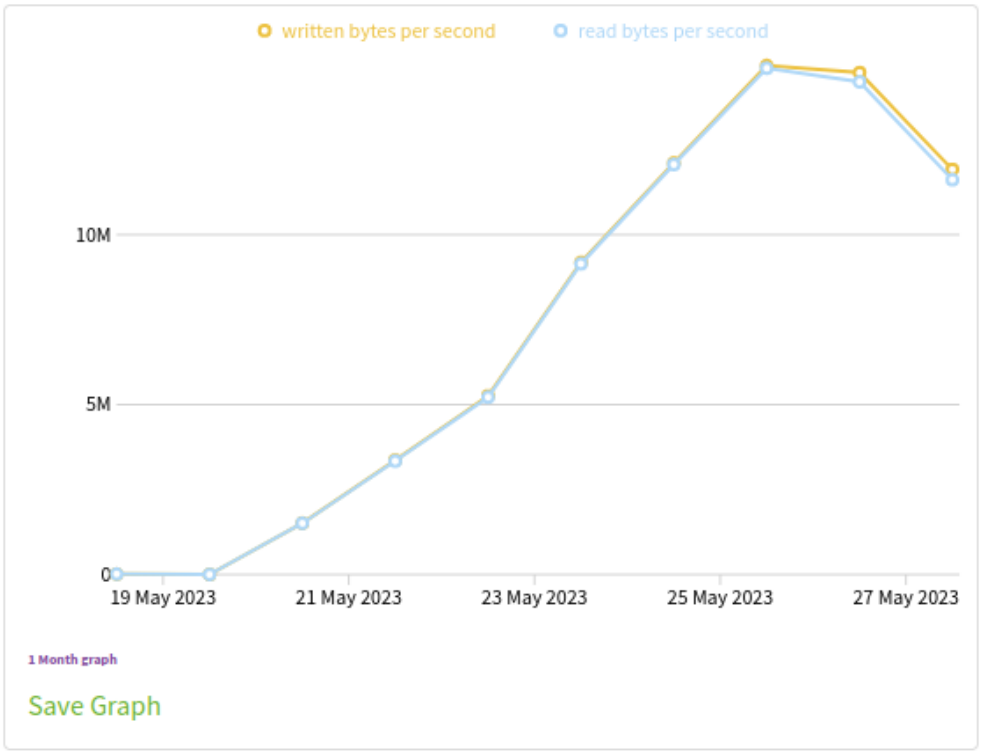

Figure B.7: Data Usage Evolution[39]

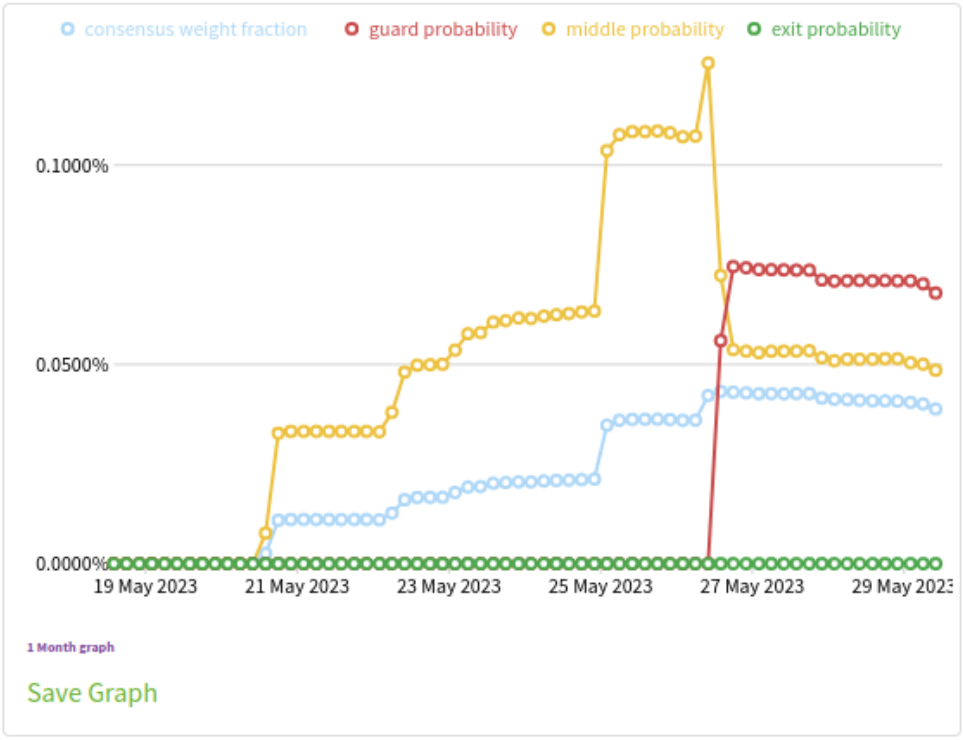

Figure B.8: Flag Assignment Evolution[39]

THS Pages C

Captcha Page

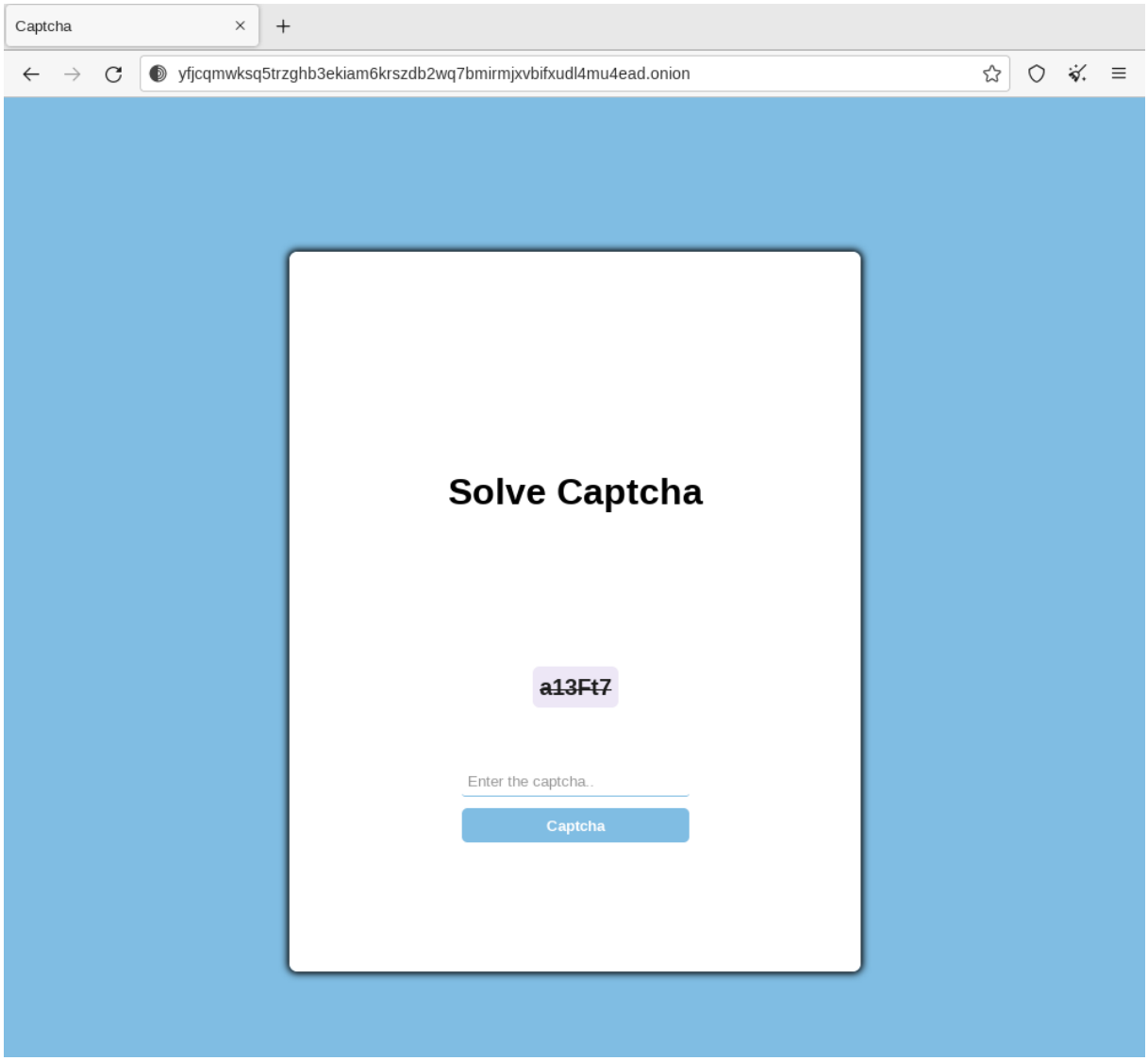

Figure C.1: Captcha Page

Home Page

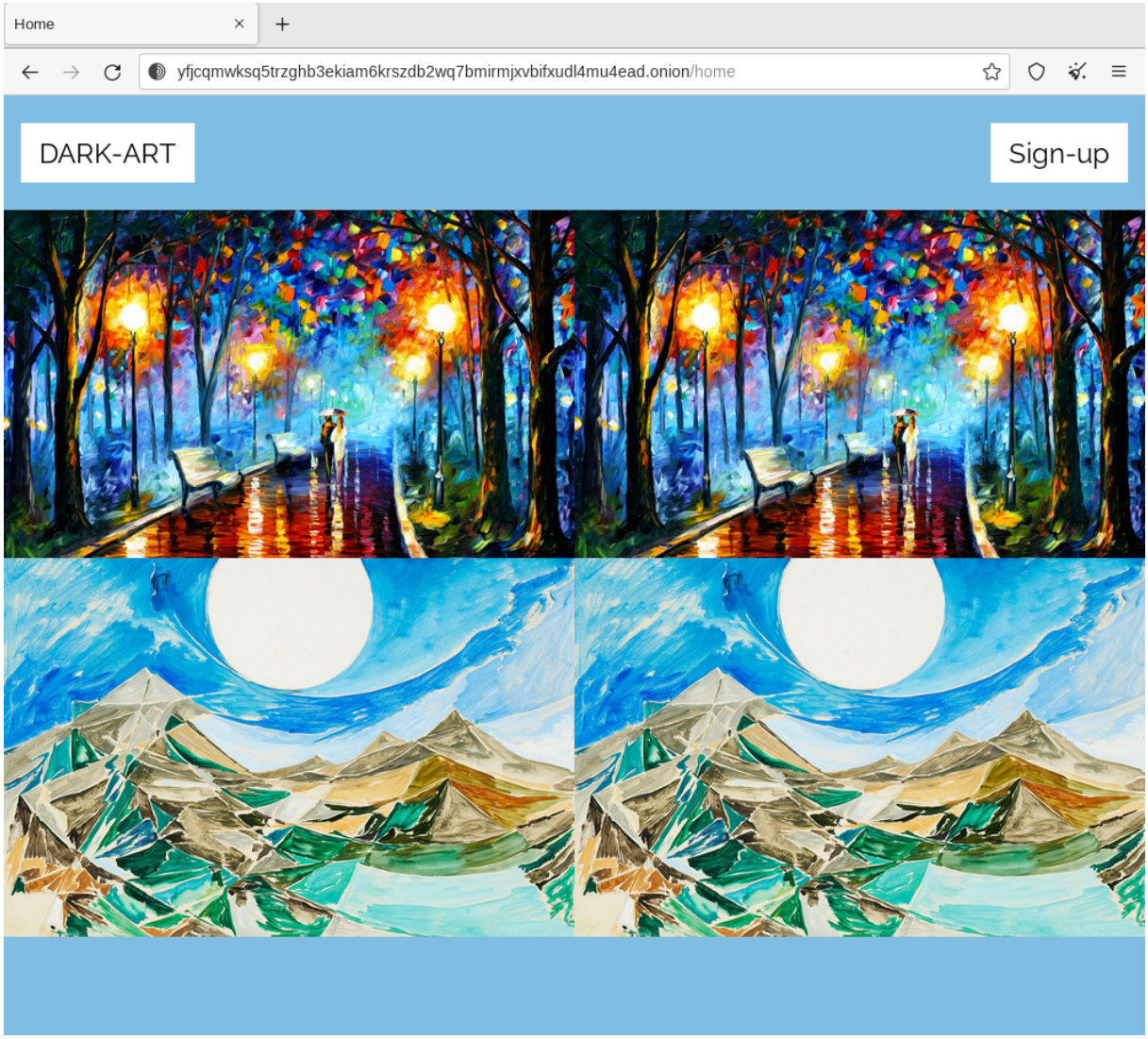

Figure C.2: Home Page

Sing-up Page

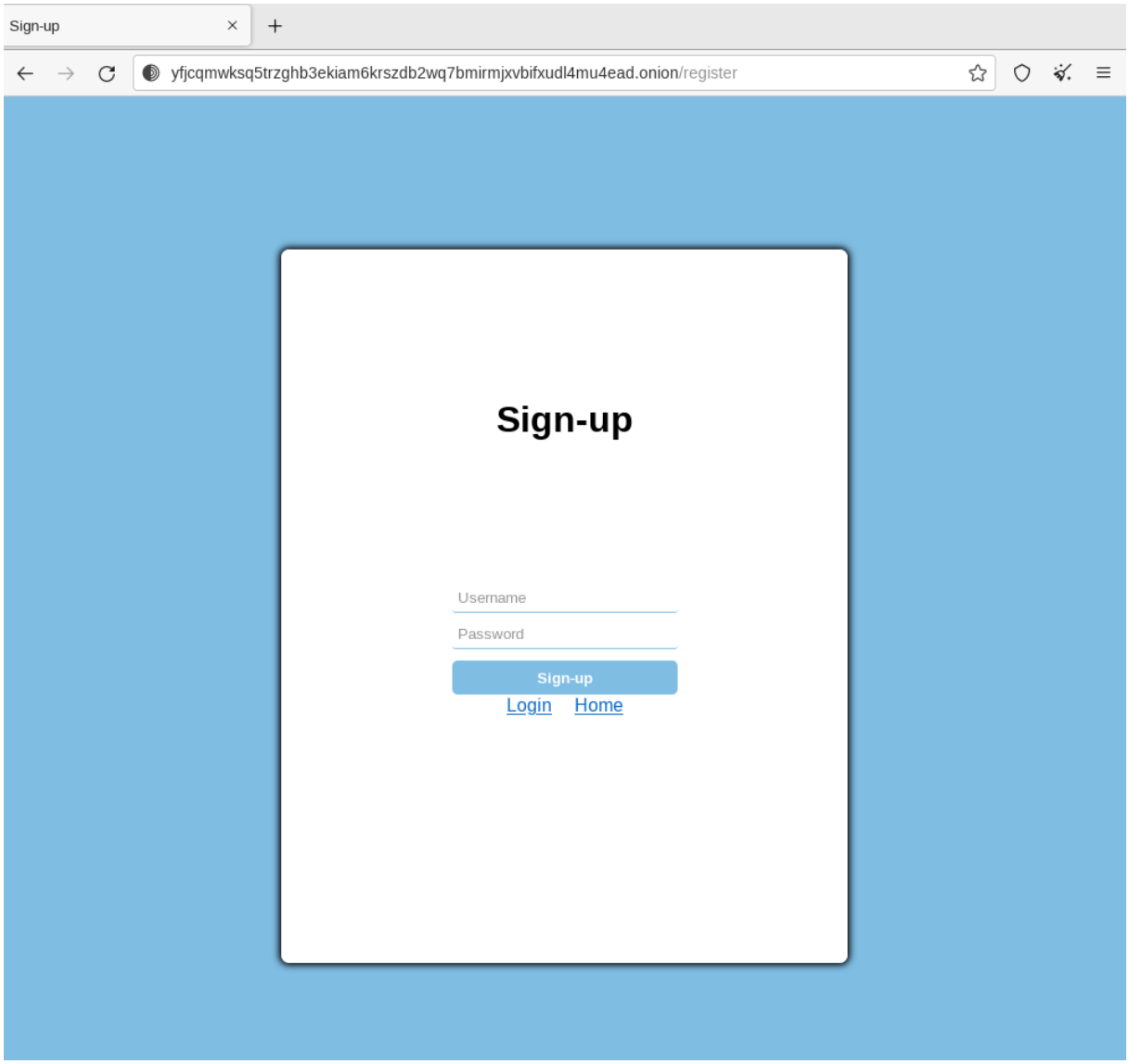

Figure C.3: Sign-up Page

Login Page

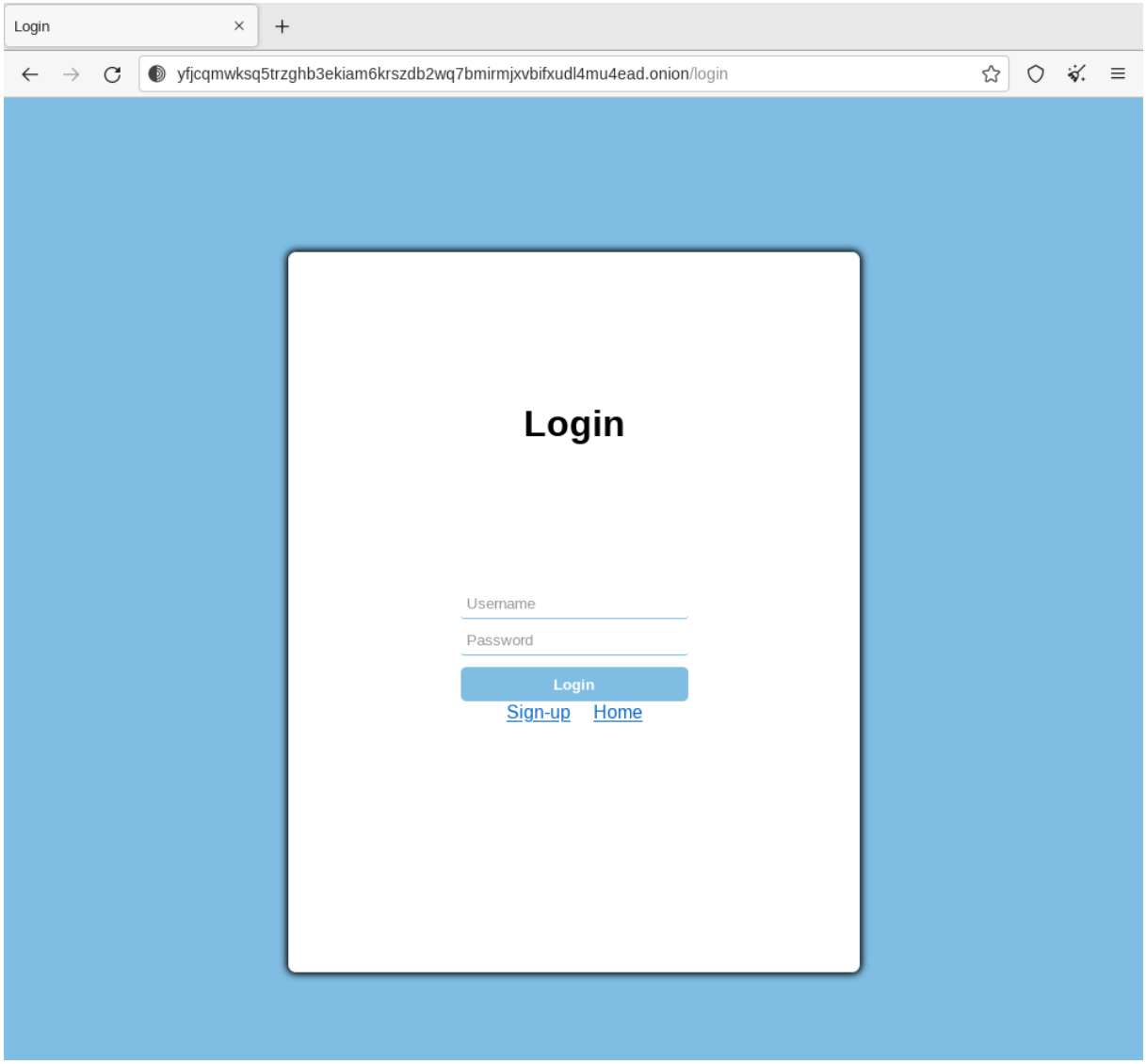

Figure C.4: Login Page

Profile Page

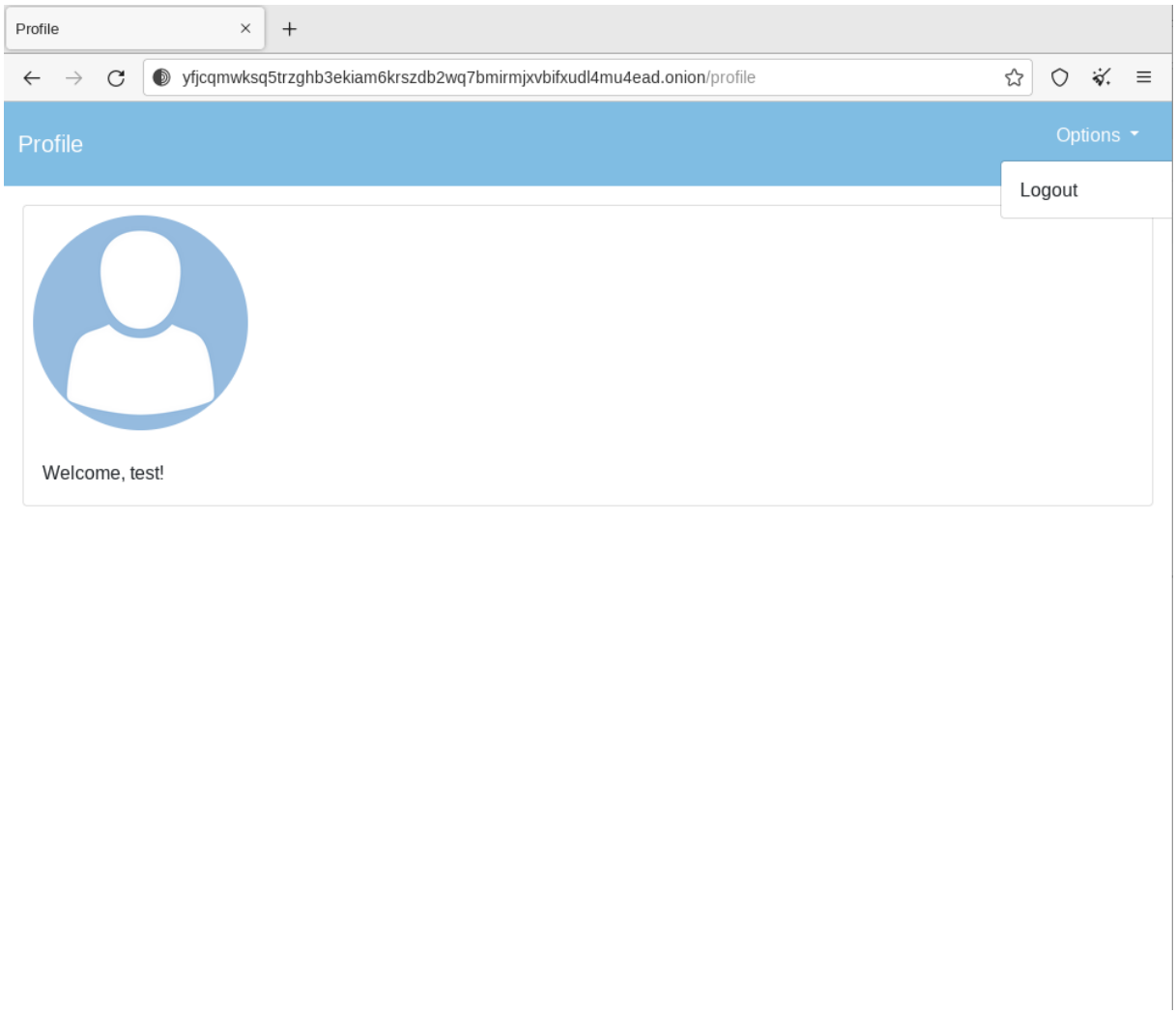

Figure C.5: Profile Page

Relay Statistics D

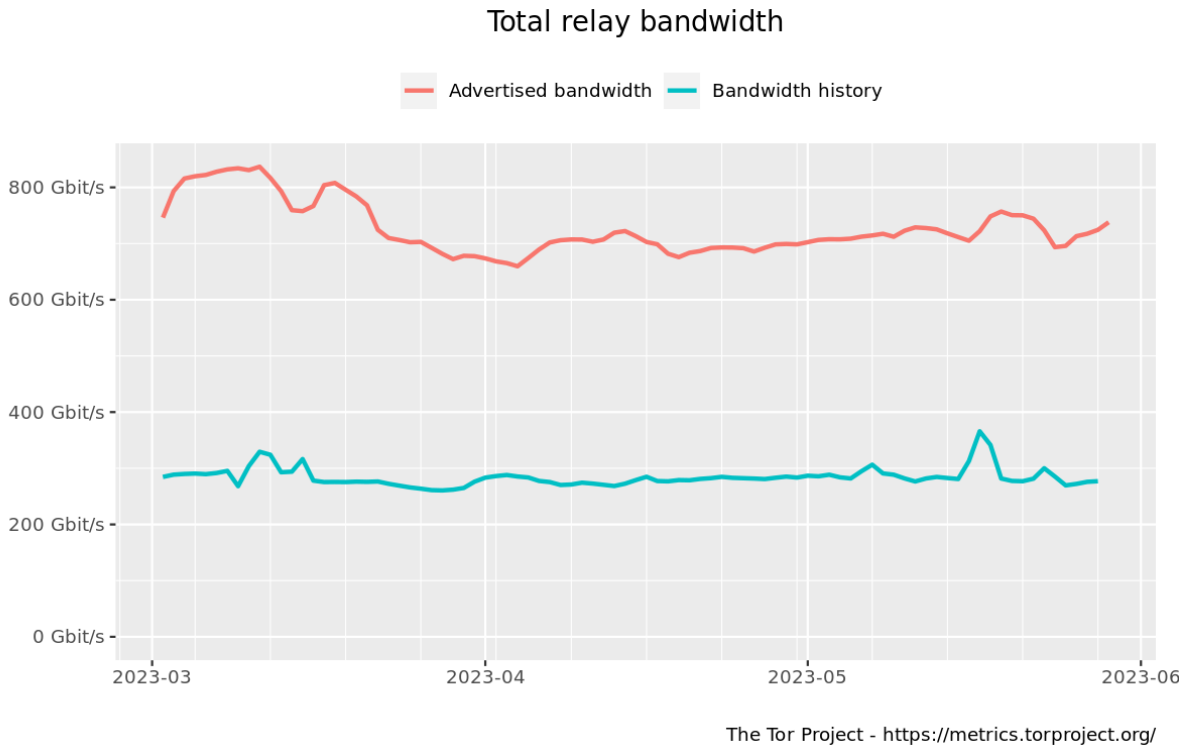

Figure D.1: Advertised vs Observed[44]

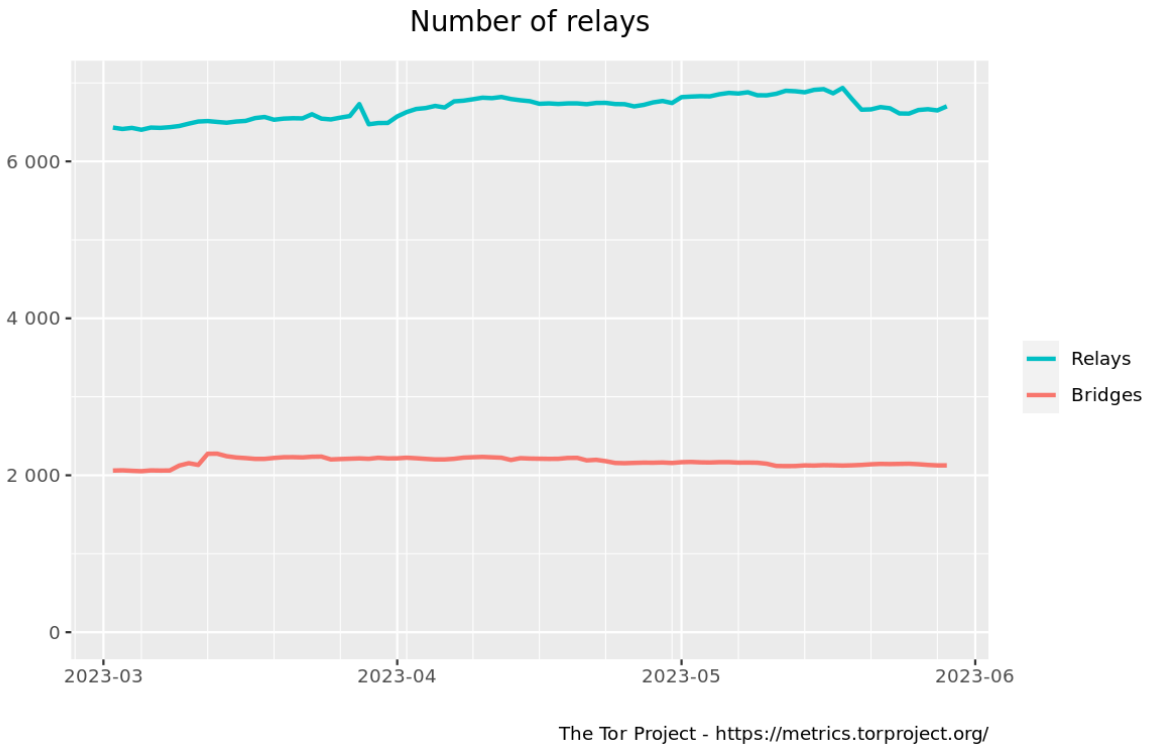

Figure D.2: Bridges vs Relays[44]

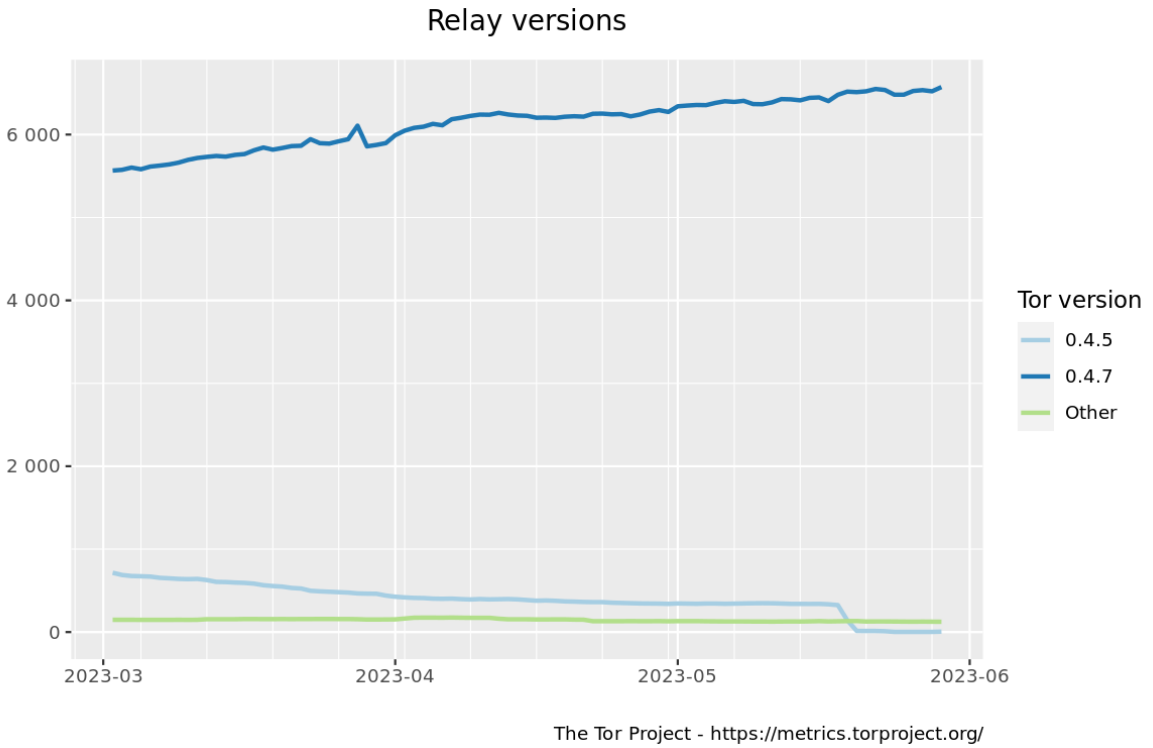

Figure D.3: Tor versions used for relays[44]

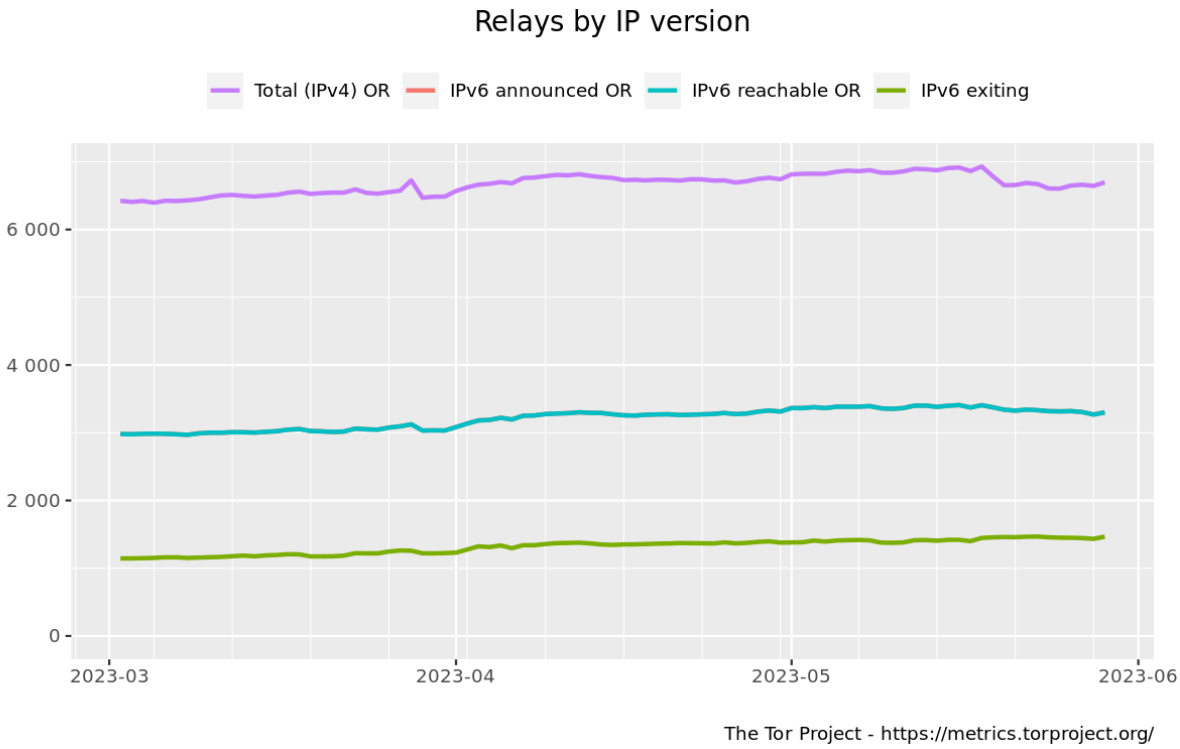

Figure D.4: IP versions in use[44]

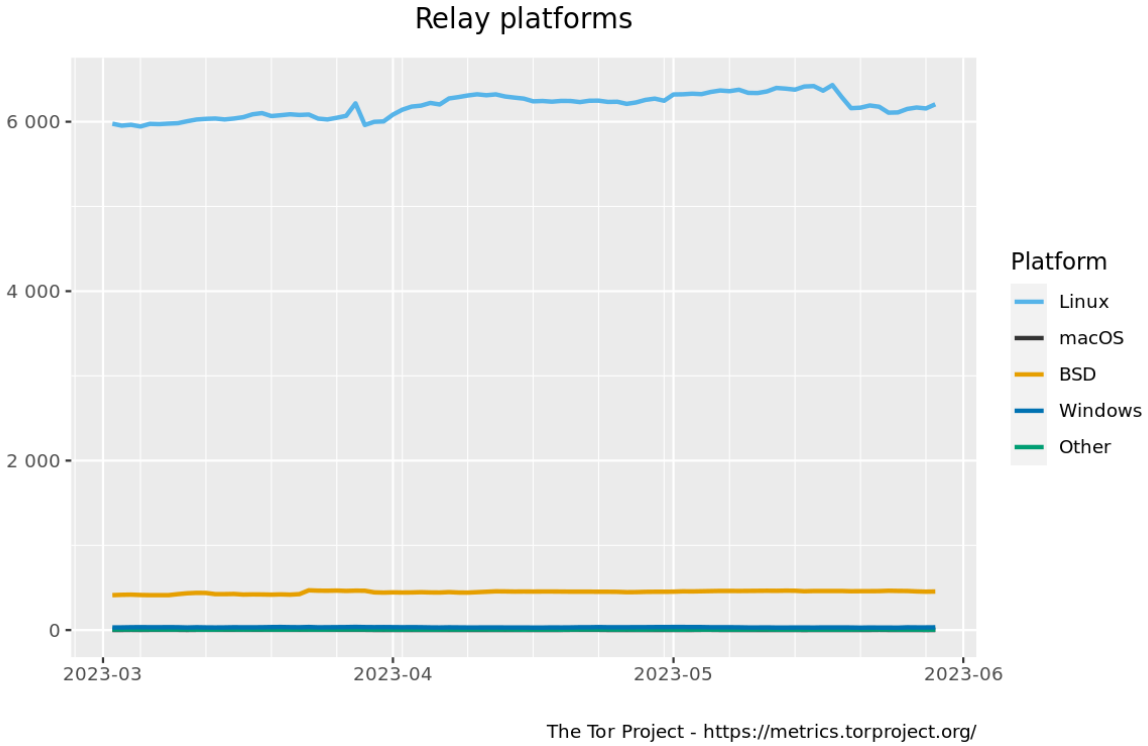

Figure D.5: Relay Deployment based on Platform[44]

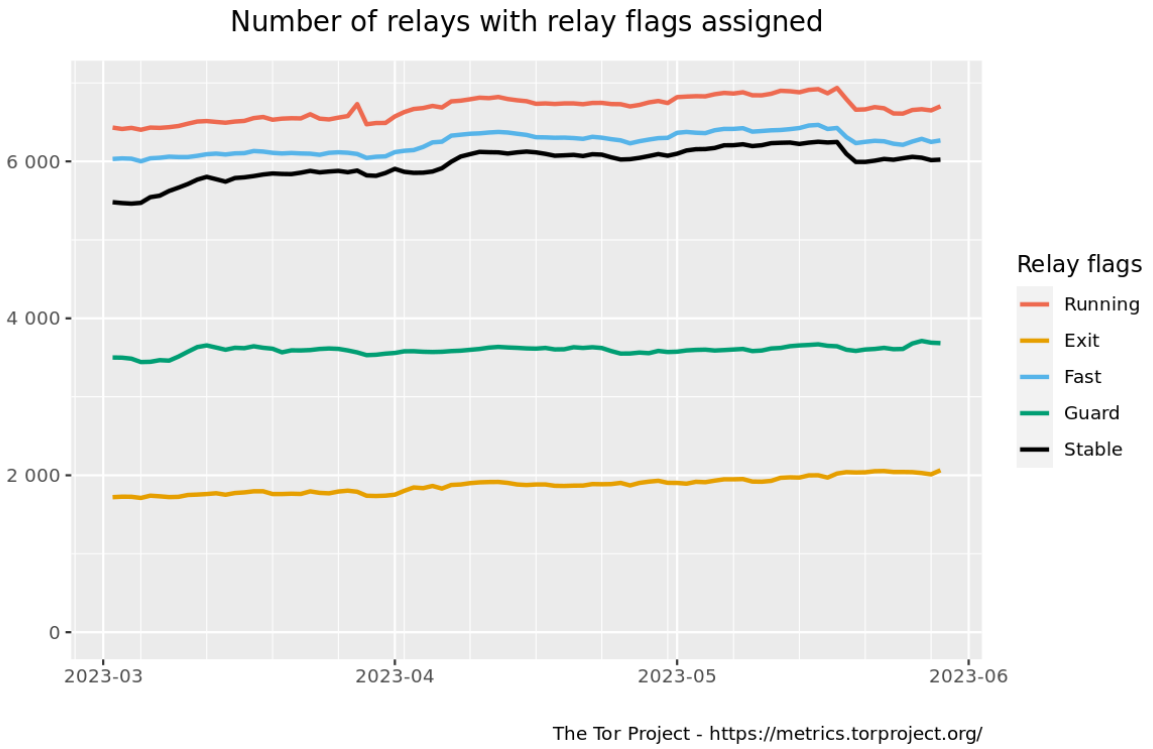

Figure D.6: Flag Types in use[44]

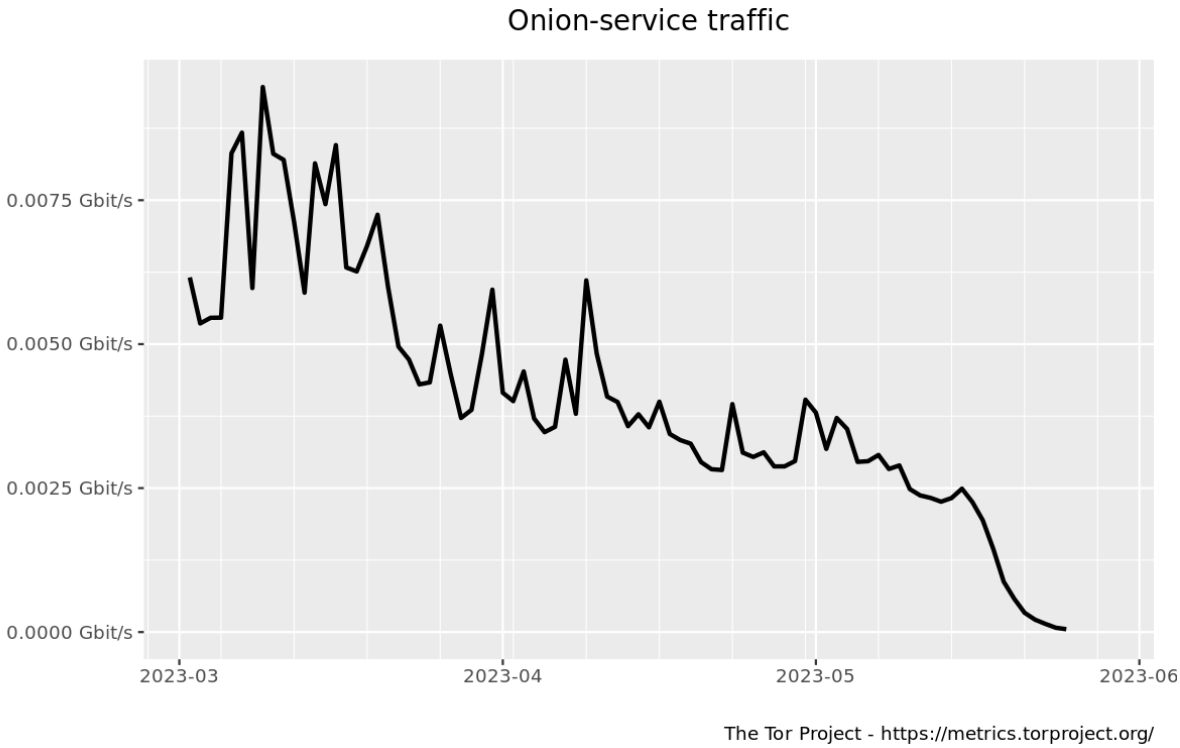

Figure D.7: .onion v2 traffic in 2023[44]

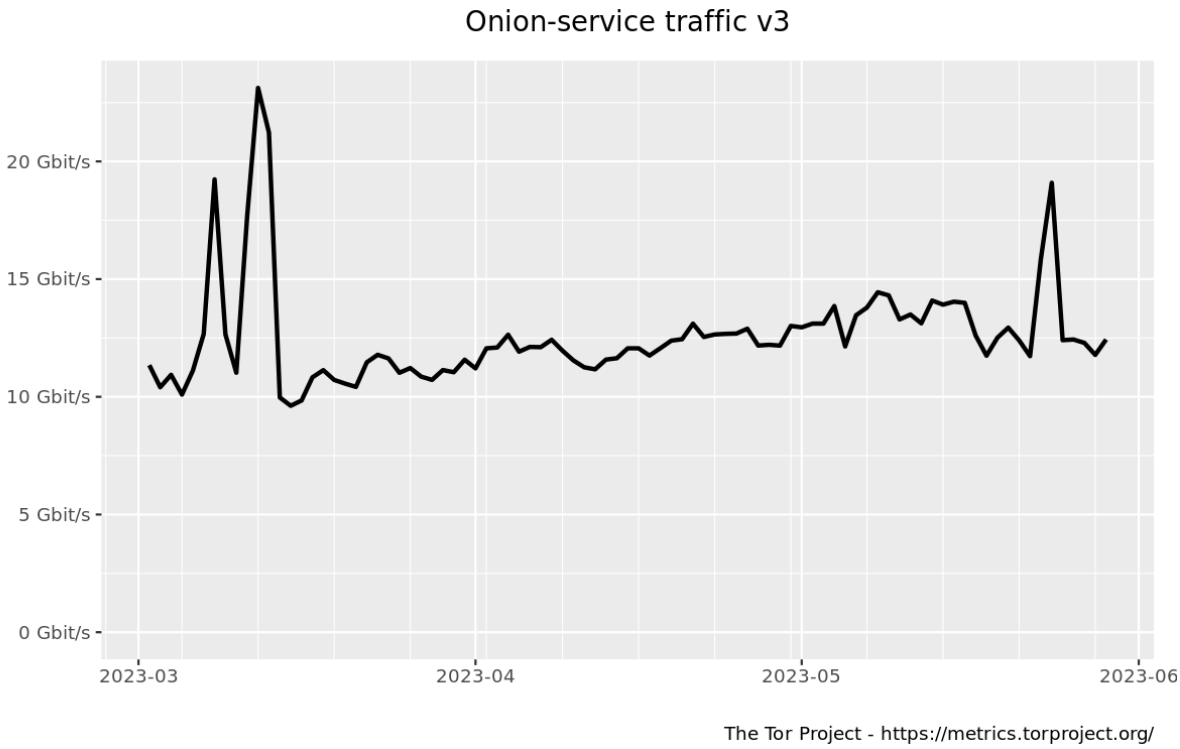

Figure D.8: .onion v3 traffic in 2023[44]

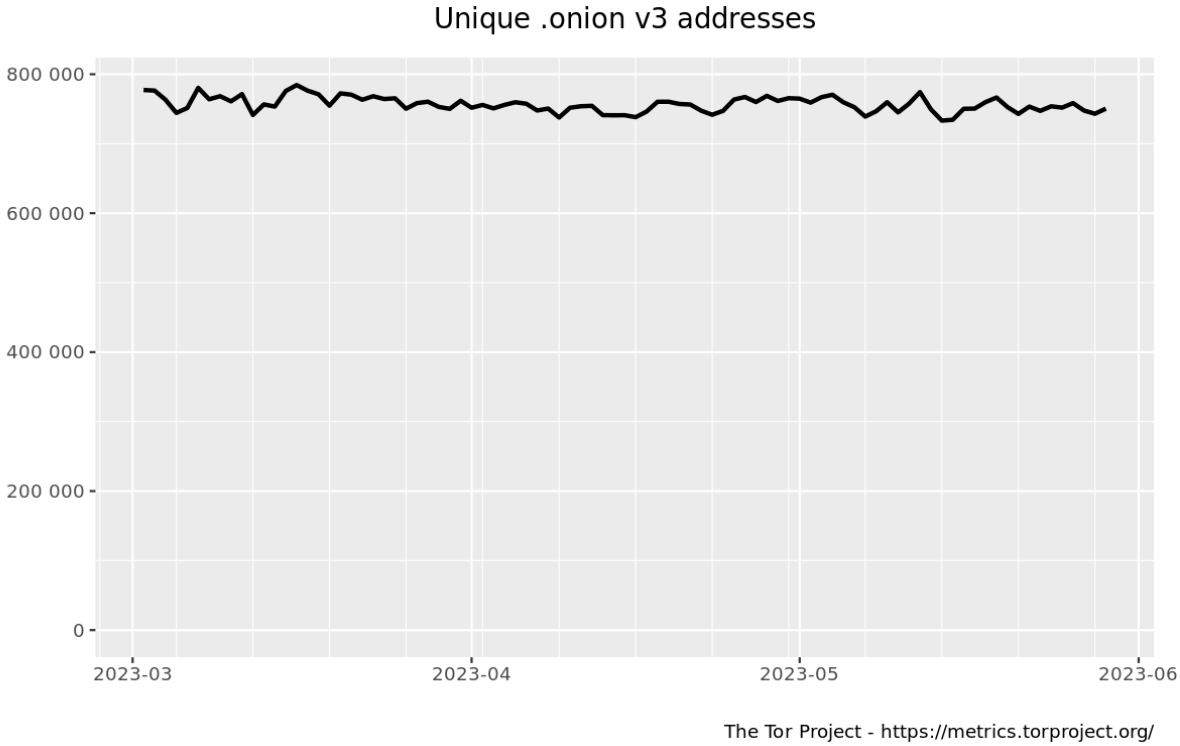

Figure D.9: Amount of .onion v3 in 2023[44]