Probing the Dark Web

Optimizing Port Scanning for Dark Web Protocol Analysis

Vyshnavi Molakala Narasimhalu

Probing the Dark Web

Optimizing Port Scanning for Dark Web Protocol Analysis

by

Vyshnavi Molakala Narasimhalu

to obtain the degree of Master of Science
Software Technology Track
with a 4TU specialization in Cyber Security

at the Delft University of Technology
to be defended publicly on June 11, 2024 at 15:00

Student number: 5757991		
Thesis committee:	Prof. dr. G. Smaragdakis	TU Delft, Chair
	Prof. Gosai Migut	TU Delft, first core member
	Prof. Harm Griffioen	TU Delft, second core member
	Dr. Mark van Staalduinen	CFLW Cyber Strategies, external supervisor

Preface

Welcome to my master's thesis, "Probing the Dark Web," the culmination of my academic journey in Master's program in Computer Science with a specialization in Cybersecurity at the TU Delft. I present this work with great pleasure and a sense of accomplishment. This journey has been about intellectual growth and personal development. The foundation of this research was laid during my early coursework, where I developed a keen interest in Cybersecurity. Through numerous discussions with my supervisor, my focus gradually sharpened, leading to the formulation of the central questions addressed in this thesis.

I would like to express my sincere gratitude to my supervisor, Dr. Georgios Smaragdakis, for his unwavering support, insightful feedback, and continuous encouragement. His positive attitude and supportive nature have been a constant source of motivation, and I have always looked up to him as a role model. It has been an honour to complete my thesis under his supervision. I am also indebted to my external supervisor, Mark, and mentors in CFLW Cyber Strategies, Eljo, Octavio and Prof. Peter. Without you, the thesis wouldn't have been possible and would have resulted in less success. The weekly research meetings at CFLW provided me with a wealth of information and innovative ideas that continuously improved my thesis. The insightful questions posed during these weekly presentations instilled deeper critical thinking, pushing me to refine my research and approach challenges from new perspectives. I would also like to extend my gratitude to the members of my thesis committee, Harm Griffoen, Gosia Migut.

My deepest appreciation goes to my family, whose love, patience, and understanding have sustained me throughout this journey. They have always encouraged and motivated me to pursue my dreams. Especially my sisters and brothers-in-law, with their wisdom, warmth, and unwavering belief in me, have been my guiding light through challenging times. I feel incredibly lucky to have such a great family who stands by my side and supports me unconditionally. Special mention goes to my nephews, Vihaan and Aarya, and my niece, Aadhya. These little bundles of joy have been my stress busters whenever I needed a break. Their innocence and laughter have brought immense joy to my heart and provided a delightful distraction from my academic endeavours.

To my friends in India, Vandhana, Valika, Shreyas, Suraj, and Keerthana, thank you for standing by me over the years. The bonds we've nurtured across the miles have shown me how enduring and meaningful friendship can be, and I am grateful for each of you. Sometimes, the most beautiful connections are those we never anticipated. I am grateful for the unexpected friendships that blossomed along the way. Their support, kindness, and companionship have made the last few months much easier, and I cannot imagine this journey without them. Thank you all for being there for me.

Lastly, this work is dedicated to all the researchers and scholars who have come before me and whose contributions have laid the groundwork for my research. I hope this thesis adds to the growing body of knowledge in the field of Cybersecurity and inspires future research endeavours.

Abstract

The inception of onion routing in the mid-1990s, evolving into Tor (The Onion Routing) and other anonymous networks, marked a pivotal moment in the quest for internet privacy. However, the emergence of the dark web, facilitated by these networks, has also increased cybercrime activities, necessitating a critical examination of its implications and challenges. Besides, the intricate security architecture of this hidden realm creates a persistent challenge in identifying and mitigating cyber threats, fostering a landscape that demands innovative methodologies for robust cybersecurity.

This thesis addresses the research gap in the existing literature, predominantly focused on TOR v2, by investigating protocols and services operating within TOR v3 onion services. Moreover, it fills the void in the literature by proposing an optimized port-scanning methodology for comprehensive analysis. Unlike previous studies, which have only considered a small dataset of onions and a limited number of ports in their TOR v2 onion services analysis, this work proposes an optimized strategy for scanning all ports, thus providing a more thorough understanding of the network's dynamics.

Our research uncovers several critical insights into the landscape of onion services. We identified many onion services operating on non-standard ports, escaping typical web crawlers and the Tor browser, which only display HTTPS/HTTP pages. Through a comprehensive port scan of 300,000 onion services, we discovered 196 unique ports, highlighting a broad spectrum of service configurations. We categorized these services into six main types: web, Bitcoin, remote access, chat, email, file transfer, and miscellaneous, with web services being the most prevalent. Additionally, we observed that a significant number of onion services discovered in 2019 remain active, suggesting durability within the dark web. Interestingly, some services exhibited extensive port usage, with up to 35 open ports reflecting diverse functionalities or potential security vulnerabilities. These findings provide a deeper understanding of the dark web's structure and the persistence of its services.

The key findings of this research sheds light on the intricacies of the dark web's inner workings. By addressing key research questions and providing clear definitions and mechanisms, this study empowers stakeholders, such as security researchers, law enforcement, and cybersecurity professionals, to navigate the digital landscape with vigilance and develop robust defence mechanisms against emerging threats.

Contents

List of Figures

List of Tables

1 Introduction

The surge in digitalization has presented a dual challenge to web security stemming from the exponential increase in internet users. This proliferation not only broadens the attack surface but also jeopardizes user privacy. These growing concerns about internet security and surveillance led researchers at the United States Naval Research Lab (NRL) to explore the concept of onion routing in the mid-1990s [1]. The concept aimed to create internet connections, ensuring user privacy by routing traffic through multiple servers and encrypting it at each stage. This initiative transformed into Tor (The Onion Routing) during the early 2000s, signifying the rise of a privacy-centric internet solution. Alternative anonymous networks like Freenet, ZeroNet, and I2P also surfaced, collectively comprising the dark web. The evolution of the dark web brought with it a surge in cybercrime, encompassing activities ranging from child abuse and drug trafficking to the illegal sales of weapons and hacking services [2]. This has garnered the attention of cybersecurity researchers and law enforcement agencies, prompting a critical examination of the implications and challenges of the dark web.

The statistics derived from Tor Metrics provided by the Tor Project [3] indicate a comprehensive and reliable source for understanding the dynamics of the Tor network. Notably, the number of Tor users has increased from 2.5 million in 2021 to 6.25 million in 2024, as shown in Figure 1.1. The number of onion services has exhibited a consistent upward trend over the past three years, experiencing a notable surge of 50% from 0.5 million in 2021 to a total of 0.8 million in 2024, as shown in Figure 1.2. This growth is paralleled by a nearly threefold increase in onion service-related traffic. Specifically, the estimated total data throughput rose from 6 Gbit/s on October 1, 2021, to 17 Gbit/s on February 1, 2024. However, interpreting the current state of the Tor network based on these historical results is challenging. The dynamic and anonymous nature of the network, coupled with the substantial increment in traffic, makes it difficult to accurately assess the network's status. It is rather speculating than determining what kind of applications are hosted, how popular or what content they provide.

1.1. Motivation

The rapid growth of the dark web, stemming from the pursuit of user privacy and anonymity, holds positive and negative consequences for society. While it serves as a haven for those seeking privacy, it also becomes a breeding ground for cybercrime activities, including data breaches and the sale of stolen information. This duality underscores the profound impact on individuals and businesses unwittingly entangled in the complexities of the dark web. Civilians, falling prey to cybercrime activities on the dark web, contend with financial losses and emotional distress. The implications are far-reaching, particularly when data breaches expose personal information, leading to identity theft and financial fraud. The dark web's trajectory is associated with a spectrum of cybercrime activities, including data breaches, drug trafficking, and illicit markets [4]. This association accentuates the urgency to delve into the fundamental protocols governing the dark web. Understanding the diverse array of protocols and services operating within the dark web becomes paramount in this dynamic landscape. This knowledge is instrumental in navigating the dark web effectively, offering insights into its functionalities and potential vulnerabilities. By unravelling the intricacies of the protocols, we gain a nuanced understanding of the dark web's inner workings and empower cybersecurity professionals to formulate robust defence mechanisms.

While certain literature has examined Tor onion service protocols, most studies primarily focus on the Tor v2 version. Moreover, prior research has limited its scope to a small subset of ports due to the inherent

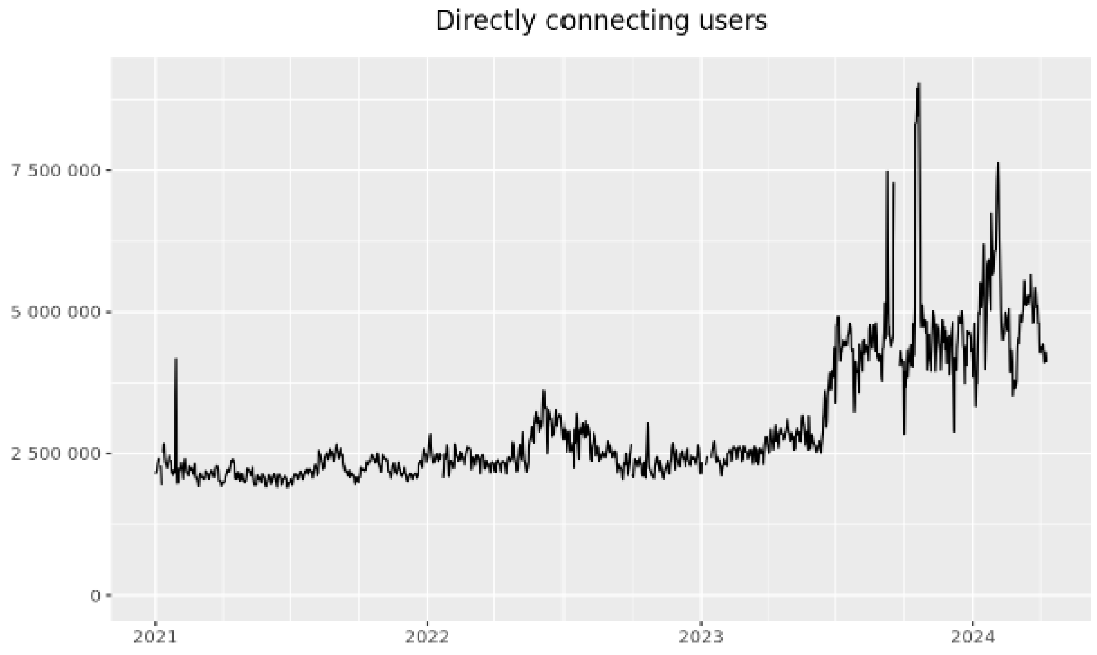


Figure 1.1: Statistics of Tor Users [3]

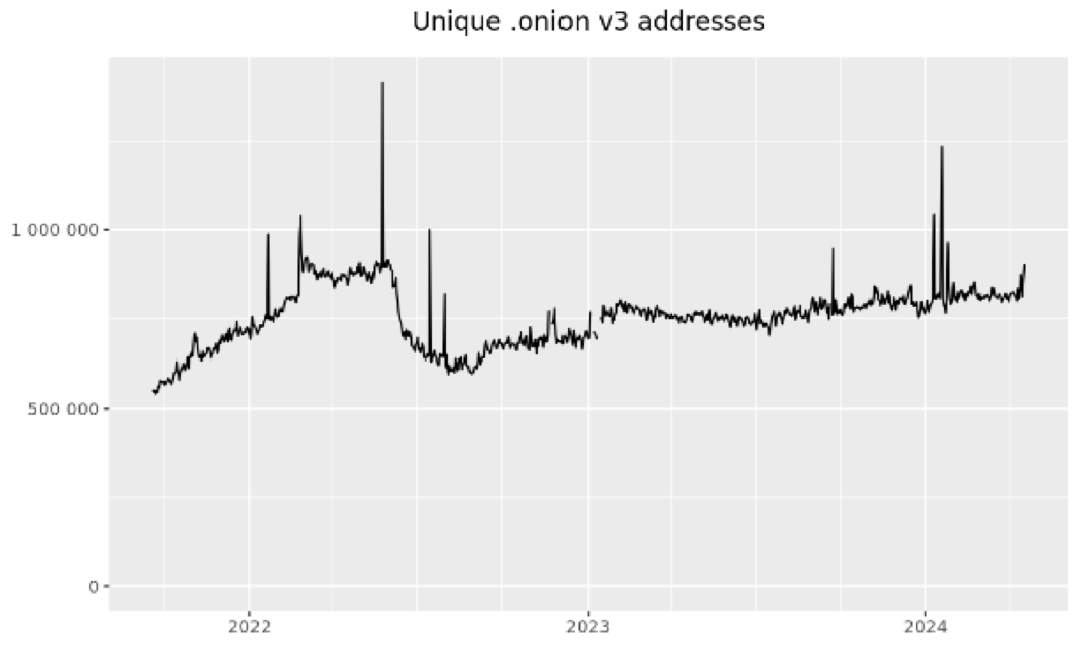


Figure 1.2: Statistics of Unique Tor v3 Onion Addresses [3]

challenges of port scanning, especially within the Tor network. This study goes beyond exploration by identifying protocols prevalent within Tor v3 onion services. Additionally, this analysis aims to broaden the range of examined protocols by providing an optimized port-scanning methodology. This methodology enables the comprehensive scanning of all possible ports, including those not previously explored in the dark web.

1.2. Stakeholders

Stakeholders in this research include:

1. Cybersecurity Professionals: By gaining insights into the intricacies of these protocols, cybersecurity professionals can refine their strategies to safeguard against potential vulnerabilities and emerging threats within the dark web.
2. Network Analysts: Network analysts play a pivotal role as stakeholders because they are tasked with optimizing network performance and identifying anomalies. An in-depth exploration of the protocols running within onion services equips network analysts with a comprehensive understanding of network behaviours, facilitating the identification of anomalies and potential security breaches.
3. Developers of Privacy-Centric Solutions: These are the developers engaged in creating software, applications, or systems that prioritize user privacy, anonymity, and security. This research empowers them to enhance the efficacy of their solutions, ensuring compatibility and adaptability to the evolving landscape of the dark web.
4. Law Enforcement Agencies: Law enforcement agencies are critical stakeholders in navigating the dark web's complex terrain. A profound understanding of the protocols utilized by Onion Services enhances its ability to monitor, track, and combat illicit activities. This knowledge aids in developing targeted strategies to counter cybercrime, ensuring a proactive approach to maintaining digital security.
5. Policymakers and Researchers: Policymakers and researchers are stakeholders seeking a nuanced comprehension of the dark web's inner workings. The insights gained from the research can guide them in formulating informed policies and strategies to address the challenges posed by the dark web. This knowledge is instrumental in crafting regulations that balance security concerns with individual privacy rights.
6. General Internet User Community: The dark web's activities indirectly impact the broader internet user community. Understanding the protocols used by Onion Services offers users awareness, enabling them to adopt informed practices for online security. This knowledge empowers individuals to navigate the digital landscape with vigilance, minimizing the risk of unintentional exposure to cyber threats.

This chapter lays the foundation for a focused exploration into the protocols and services operated by Onion Services, offering a road map for deciphering the complexities of the dark web's hidden network.

1.3. Research Questions

This research aims to investigate and provide insights regarding prevalent protocols on the Dark Web, optimize port scanning strategies, and establish correlations between Dark Web infrastructures. To achieve this objective, this thesis addresses the following research question (RQ) and their respective sub-questions (SQ):

Research Question

What prevalent protocols are utilized for communication on the Dark Web, and what types of activities occur beyond traditional web protocols such as HTTP or HTTPS?

SQ1 **How can we develop a specialized port scanning tool tailored specifically for Dark Web environments?** Port scanning in the Dark Web poses distinct challenges, primarily due to its decentralized architecture, multiple layers of encryption and the implementation of anonymity-preserving technologies. These features obscure the true origins and destinations of network traffic, complicating traditional scanning methods. Additionally, the encryption layers within Tor further hinder port scanning efforts, resulting in slow and inefficient scans. However, developing an advanced tool to identify open ports could significantly streamline the process for law enforcement agencies and researchers. While existing tools like Nmap, Netcat, and Ncat are commonly used for Dark Web port scanning, their slow performance renders them impractical for comprehensive scans [5]. Investing in specialized port scanning tools tailored for Dark Web environments can enhance our understanding of Dark Web infrastructures and enable law enforcement agencies and researchers to allocate their resources more effectively toward investigative efforts. Hence, this sub-question aims to design and implement a tool capable of effectively scanning Dark Web networks, considering their unique architecture, communication protocols, and stealth requirements.

SQ2 **What are the key characteristics and dynamics of the onion services landscape within the Dark Web?** One significant challenge arises from the transient nature of services within the Dark Web, where they appear and disappear rapidly, making it difficult to maintain an accurate and up-to-date list of targets for scanning [6]. This sub-question aims to understand the onion services dynamics within the Dark Web. Investigating the dynamics of the onion services landscape involves a comprehensive analysis of the various factors and processes contributing to its ever-evolving nature. We aim to investigate onions' availability and categorise onions based on their availability - offline / online. This encompasses the continuous shifts and transformations observed in the activity and availability of onion services. Fluctuations in the number and variety of onion services, the emergence of new ones and the disappearance of existing ones indicate the dynamic nature of the Dark Web ecosystem. This sub-question mainly deals with finding the status of onions and their corresponding response times and monitoring these onions to see if any interesting trends emerge.

SQ3 **How do port scanning activities contribute to uncovering the prevalent protocols within the Dark Web ecosystem** In this sub-question, we shift our focus from finding the status of onions to port scanning. Here, the aim is to scan open ports on the onions that are available or online. This includes identifying key characteristics such as the types of services offered, the prevalence of different protocols, and the distribution of services across various domains and communities. Researchers can detect emerging trends and developments within the Dark Web ecosystem by monitoring changes in the onion services landscape. This may include the emergence of new types of services, shifts in the popularity of certain protocols, the evolution of onion services, or changes in the behaviour of Dark Web users and communities.

SQ4 **How can we optimize traditional port-scanning methods to suit the challenges of Dark Web networks better suit the challenges of Dark Web networks better?** The dynamic and ever-evolving landscape of the Dark Web necessitates continual monitoring and adaptation of scanning strategies to keep pace with environmental changes. Resource constraints such as bandwidth and access restrictions further exacerbate the challenges, impeding the efficacy and scalability of port scanning endeavours. Additionally, conventional port-scanning techniques often lack comprehensive coverage of the Dark Web due to slow tool performance and limited port range coverage. Moreover, conducting a full scan is impractical and time-consuming, hindering researchers and law enforcement agencies from understanding the onion services landscape. To bridge this gap, this research capitalizes on a comprehensive dataset of onion services collected over four years by Dark Web Monitor, aiming to optimize traditional port-scanning methods.

1.4. Contributions

This thesis makes notable contributions to the Tor Network Protocols Landscape, outlined as follows:

C1 Diverging from the predominant focus on Tor v2 in existing literature, this research centres on the latest v3 version of Tor onion services, providing insights into the advancements and changes in the protocol.

C2 An extensive Protocol Analysis of Tor v3 onion services is conducted, encompassing the examination of the full range of ports (65,536). This comprehensive approach ensures a thorough understanding of the network's intricacies and potential vulnerabilities.

C3 Clear definitions and mechanisms are elucidated for detecting the operational status of an onion service—whether it is offline or online. This clarity contributes to a more precise understanding of the network's dynamics. Here, we transition from the traditional web-based view of the dark web to any application-based view.

C4 The port scanning tool developed for this thesis significantly enhances the monitoring and analysis of onion services on the Dark Web. It incorporates functionalities for categorizing online, offline, or temporarily unavailable (T.U) onion services, with a continuous monitoring queue system. Utilizing goroutines in Golang (Go) enables concurrent execution of port scans, optimizing performance and efficiency.

C5 An optimal port-scanning methodology is presented, offering a refined and efficient approach for exploring and assessing the security landscape of Tor onion services. This methodology aims to enhance the effectiveness of port-scanning procedures within the Tor network.

1.5. Outline

The report is structured as follows. Chapter 2 summarizes the theoretical groundwork, covering aspects of port scanning, onion routing, and the Tor onion service mechanism. An overview of the current literature on port scanning and protocol analysis in the Dark Web is presented in Chapter 3. Chapter 4 provides a detailed overview of the comprehensive dataset utilized in this research, sourced from CFLW Cyber Strategies' Dark Web Monitor (DWM). Chapter 5 details the methodology used to conduct this research, including the architecture of the port scanning tool, validation of the tool, and the algorithm used to optimize port scanning. Research Findings in Chapter 7 are followed by the reflections on the data analysis and limitations in Chapter 8. Lastly, Chapter 9 provides a synthesis of the onion services landscape based on the insights discovered and proposes future research directions.

2 Background

This chapter provides the foundational knowledge required to follow the research. First, we explain the infrastructure of the World Wide Web - Surface, Deep, and Dark Web. Subsequently, we delve into the intricacies of the Tor network and the Tor Onion Service (OS) mechanism. Its purpose is to enhance our understanding of the advantages and challenges posed by the Tor network in its pursuit of anonymity and security. This exploration draws a clear contrast with the more straightforward dynamics of the surface web. Finally, port scanning concepts are explained in detail.

2.1. World Wide Web Infrastructure

The Internet is a decentralized network that allows computers worldwide to communicate with each other through standardized protocols. It functions as the infrastructure that enables diverse services, including email, file sharing, online gaming, and, most prominently, the World Wide Web (WWW).

The Internet provides connectivity and networking capabilities, while the WWW is a model built upon the Internet's infrastructure, allowing users to browse, access, and share information using web browsers. Search engines are pivotal in navigating and retrieving information within the WWW. They serve as the primary gateway for users to access information on the Internet. The WWW relies on the Internet, and search engines contribute to the accessibility and usability of the information hosted on the WWW.

In the vast expanse of the Internet, distinct layers of web accessibility unfold, each with unique characteristics. The triad - Surface, Deep, and Dark Web is illustrated in Figure 2.1. Beyond the familiar websites of the Surface Web lies the concealed Deep Web and, within it, the mysterious Dark Web.

1. **Surface Web** is the part of the web that is indexed by traditional search engines. It comprises static web pages easily accessible to the general public through standard web browsers such as Google Chrome, Internet Explorer, and Mozilla Firefox[8].
2. **Deep Web** contrasts the easily accessible Surface Web. The Deep Web lies beneath the surface, analogous to the submerged portion of an iceberg. This hidden realm comprises a significant portion of the World Wide Web that traditional search engines cannot index, making it hidden from conventional search queries. The Deep Web contains a vast array of information, including private databases, dynamic web pages, non-HTML, non-contextual or non-scripted content, private sites (password-protected websites), and limited-access networks (darknets or sites hosted on infrastructures that require the use of specific software like Tor to access). Its significance lies in providing a secure space for data that requires controlled access, protecting it from indiscriminate online searches [8].

   For example, consider subscription-based streaming services like Netflix. When users visit the Netflix homepage (www.netflix.com), they are on the Surface Web. Here, they can see the catalogue, featured shows, and general information that search engines can index. If someone searched for "Stranger Things" on Google and clicked on the Netflix link in the search results, that interaction would be part of the Surface Web. Once a user subscribes to Netflix and logs in, they enter the Deep Web of Netflix's content. Users' interactions within the platform, such as browsing categories, adding titles to their watchlist, or downloading movies, generate non-indexed Deep Web content.
3. **Dark Web** is part of the deep web that is intentionally hidden and inaccessible through standard web browsers despite its public availability. Thus, the Dark Web can be defined as a segment of the

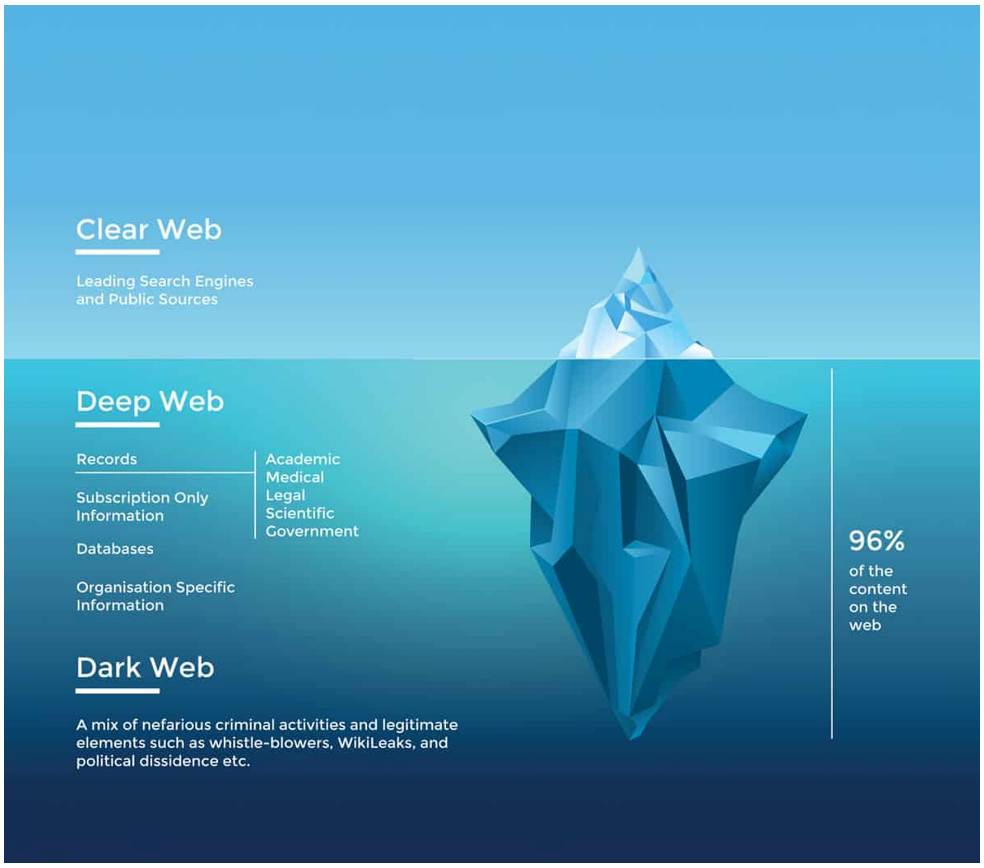

Clear Web
Leading Search Engines and Public Sources
Deep Web
Records
Subscription Only Information
Databases
Organisation Specific Information
Academic
Medical
Legal
Scientific
Government
96% of the content on the web
Dark Web
A mix of nefarious criminal activities and legitimate elements such as whistle-blowers, WikiLeaks, and political dissidence etc.

Figure 2.1: World Wide Web Infrastructure, an analogy with an iceberg. Retrieved from [7]

Internet, specifically a subset of the Deep Web, requiring specialized software for access. Darknets, the interconnected networks forming the Dark Web, differ from the broader World Wide Web due to their unique routing, encryption, handshaking, and data exchange protocols, ensuring anonymity and secure communication. Darknets constitute the infrastructure of the dark web, creating private networks where connections are established only among trusted peers [8]. This thesis is specifically centred on probing the Tor network.

Notably, the Surface Web accounts for a mere 4% of the internet, while the vast majority—96%—resides in the concealed Deep Web. Within this hidden layer, the enigmatic Dark Web constitutes 6% [9]. This prevalence of content on Deep Weeb reflects the proliferation of web applications tailored to individual users and their associated accounts. Conversely, the Dark Web lies at the depths of the ocean, accessible solely through dedicated software. Unlike conventional Internet protocols, the Dark Web operates on a customized framework, necessitating specialized programs to navigate its unique architecture. While various software types coexist within the Dark Web ecosystem, this document will focus primarily on the most commonly used platforms. Before delving into specific applications, however, a comprehensive exploration of the Dark Web's foundational aspects is essential.

2.2. Dark Web

Dark Web aimed to establish a platform where users could navigate content privately, anonymously, and securely shielded from surveillance by government or entities capable of monitoring online activities. This feature made the Dark Web particularly appealing to individuals who sought to exchange ideas and opinions without fear of public exposure. However, with many innovations, users recognized the profit potential and exploited the platform for unethical purposes.

The phenomenon of criminal activity on the Dark Web presents an intriguing area of study within cybercrime. Understanding the motivations driving individuals to engage in illicit behaviour, whether driven

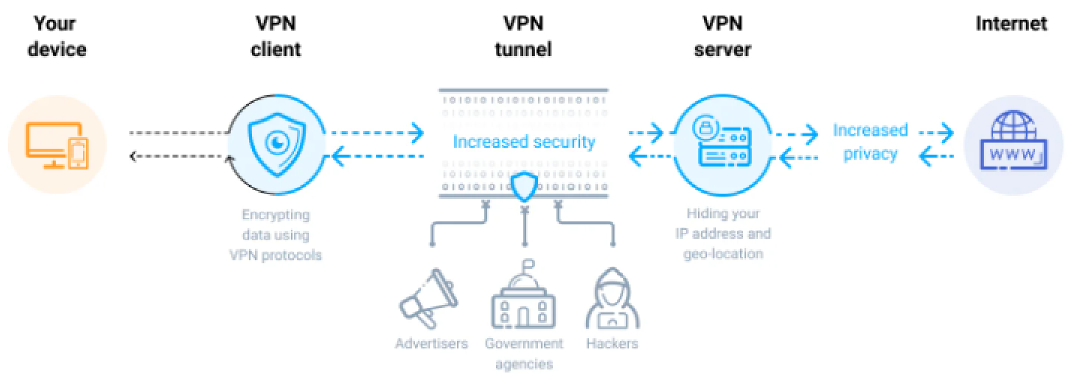


Figure 2.2: L2TP/IPsec VPN Protocol [11]

by financial gain or personal gratification, remains a subject of limited research. Exploring parallels between criminal groups operating on the Dark Web and traditional offline criminal organizations could provide valuable insights. Theoretical frameworks from criminology, such as social disorganization theory, suggest that factors like heterogeneity, instability, and decentralization within the Dark Web ecosystem foster an environment conducive to illicit activities [10].

Alternative solutions for achieving anonymity and secure communication include using Virtual Private Networks (VPNs). VPNs facilitate communication across different protocols and enable users to conceal their IP addresses or operate within specific network environments, such as enterprise networks. VPNs use a tunnelling protocol to establish a secure connection between the user's device and the VPN server. Common tunnelling protocols include OpenVPN, IPSec, L2TP/IPSec, and PPTP. These protocols encapsulate the user's data packets within encrypted packets, ensuring that data transmitted over the internet remains secure and private.

Let's say Alice wants to browse the internet securely and privately. Without a VPN, her IP address (e.g., 203.0.113.1) would be visible to websites she visits, her ISP, and potentially other parties monitoring her internet traffic. With a VPN, Alice first connects to a VPN server in a different geographical location, let's say in the United States, before accessing the internet. The VPN server acts as an intermediary between Alice and the internet, encrypting her traffic and routing it through the VPN server. After connecting to the VPN server, Alice's IP address appears to change to the IP address of the VPN server (e.g., 198.51.100.1). When Alice visits a website, the website's server sees the IP address of the VPN server (198.51.100.1) instead of Alice's original IP address. Alice's ISP only sees encrypted traffic between her device and the VPN server, but it cannot see the specific websites she visits or the data exchanged. This is clearly demonstrated in Figure 2.2

However, VPNs have inherent limitations. While they offer protection to users by obfuscating their origin, they do not extend the same level of anonymity to the servers themselves. Users must place trust in the VPN server owners, who have visibility into the connections passing through their servers. This reliance introduces potential vulnerabilities, as server owners could compromise user privacy by monitoring connections or exposing historical logs. Instances of such breaches have occurred in the past, with perpetrators often selling the compromised data on the Dark Web, underscoring the complex interplay between security, privacy, and trust in the digital landscape.

Among the various networks operating within the Dark Web, three prominent ones stand out: Freenet, I2P, and the Tor Project. However, this research will focus solely on the Tor Project, which emerged as the pioneering darknet network at the turn of the 21st century and remains the most widely used and influential darknet today. The Tor Project revolutionized the concept of the Dark Web, fundamentally altering its definition and reshaping the landscape of anonymous online communication.

2.3. The Onion Router

The Tor (The Onion Router) network represents a critical component within the landscape of secure and anonymous communication on the internet. Tor's resilience lies in its decentralized nature, as the network relies on a multitude of volunteer-operated relay nodes scattered across the globe. This enhances the network's resistance to censorship and mitigates the risk of single points of failure.

In mainstream web browsers like Internet Explorer or Google Chrome, we connect to web servers via TCP connections on port 80. Through these TCP connections, data packets are exchanged and the client's IP address are exchanged (encrypted packets if SSL/TLS protocols are in use). Consequently, internet service providers (ISPs) can easily discern the identities of users and their communication partners. However, it becomes possible to obfuscate client identification information, including the IP address, when utilising the Tor browser. Moreover, Tor enables servers to maintain anonymity through what are referred to as hidden services. Before delving into the concepts of the Tor Project, it is essential to understand its foundational technology, known as onion routing.

2.3.1. Onion Routing

Onion routing is a technique used to achieve anonymous communication over a public network such as the Internet. It encrypts data multiple times, each time adding a layer of encryption (like the layers of an onion), hence the name "onion routing." This process helps to obfuscate the origin and destination of the data, enhancing privacy and anonymity. The concept of onion routing facilitates anonymity by ensuring that no single entity along the communication path can determine the data's sender and receiver. This is achieved through a series of intermediary nodes, known as onion routers or onion relays, which forward data packets in a layered manner.

A comprehensive way to illustrate onion routing is by envisioning a scenario where a client seeks to communicate with a web server, perhaps to retrieve a web page as shown in Figure 2.3. Initially, the client must create a route consisting of onion routers (by default 3), which will serve as intermediaries for transmitting the data between the server and the client. Before we explain the example, we need to understand a few concepts:

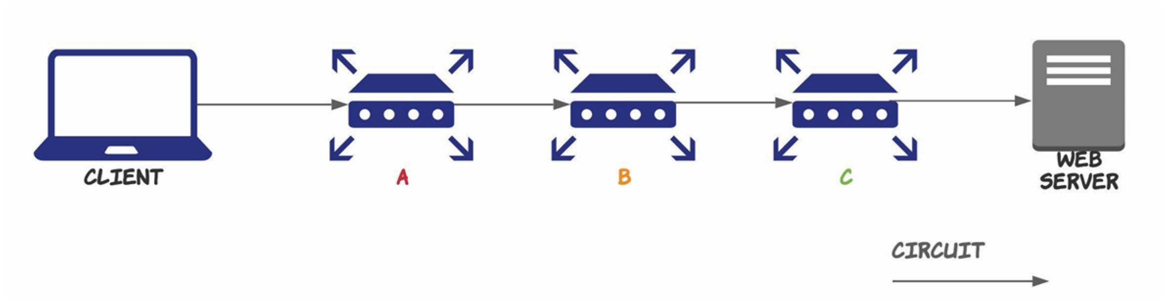


Figure 2.3: Scenario when a client seeks to communicate with a web server through Tor

2.3.1.1. Cell Structures

Cell structures serve as the fundamental units of data exchange, encapsulating various types of information essential for circuit establishment, data transmission, and network management. They play a pivotal role in facilitating secure communication between the client and the intermediary routers within the Tor network. Data packets are structured into fixed-size cells, typically 512 bytes in size. Each cell consists of a header and a payload. The header of a cell typically contains a circuit Identifier (circID): This field uniquely identifies the circuit associated with the cell, allowing routers to route and process incoming cells accurately. Circuit identifiers are crucial for distinguishing between multiple circuits traversing the same routers concurrently. Command Code: The command code specifies the purpose or action to be performed by the recipient router upon receiving the cell. Common command codes include CREATE, EXTEND, RELAY, DESTROY, and PADDING, each serving distinct functions within the Tor protocol. The payload contains the actual data to be transmitted. Based on the command code, the cells can act as a control or relay cell whose structure is shown in Figure 2.4.

Control cells are used for managing and controlling the behavior of nodes within the Tor network.

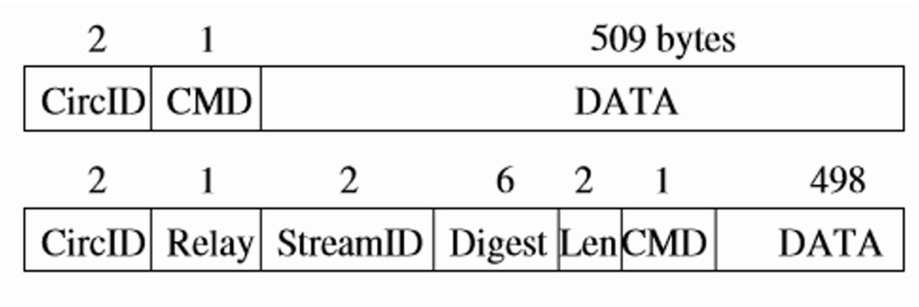


Figure 2.4: Control cell and Relay cell structures. Retrieved from [12]

Control cells are exchanged between Tor nodes and the Tor control port, allowing external entities (e.g., client applications or network administrators) to interact with the Tor network. Key fields in a control cell:

- Command: Specifies the type of control operation to be performed, such as circuit creation, circuit teardown, or network status query.
- Circuit ID: Identifies the circuit to which the control operation applies, allowing for targeted circuit management.
- Payload: Contains additional data required for the specified control operation, such as parameters for circuit creation or status information for network queries.

Relay cells are crucial components within the Tor network responsible for carrying end-to-end stream data between nodes. These cells feature an additional header, the relay header, positioned at the forefront of the payload. Relay cells are used for both data relays between nodes (e.g., from client to entry node, from relay to relay) and for returning data back along the circuit. Key fields in a relay cell:

- Command: Relay commands embedded within the header dictate various operations within the network, such as relaying data downstream, initiating stream connections (relay begin), gracefully closing streams (relay end), handling broken connections (relay teardown), and notifying the originator of successful stream openings (relay connected). Additionally, commands like relay extend and relay extended facilitate circuit extension by a hop and acknowledge such extensions. In contrast, relay truncate and relay truncated are used for partial circuit teardown and acknowledgement.
- Stream ID: Identifies the data stream associated with the relay cell, allowing multiplexing streams over the same circuit.
- Digest: An end-to-end checksum that provides integrity protection by including a cryptographic hash of the payload to detect tampering or corruption.
- Length: Specifies the length of the relay payload.
- Payload: Contains the actual data being relayed, encrypted multiple times with the encryption keys of each successive relay in the circuit.

2.3.1.2. Circuit Establishment

The client initiates the circuit establishment process by selecting a series of intermediary nodes, known as onion routers, to form the circuit. It negotiates encrypted connections with each selected router to establish secure communication channels. During the negotiation phase, the client and each router perform key exchange using asymmetric encryption techniques such as Diffie-Hellman key exchange. Through this key exchange process, the client and routers agree on symmetric encryption keys that will be used to encrypt and decrypt data within the circuit.

Once the negotiation is complete, the client sends CREATE cells to the first selected router, known as the entry node, to request the establishment of the circuit. The entry node processes the CREATE cell and selects the next router in the circuit, known as the middle node, to extend the circuit. The entry node then sends an EXTEND cell to the middle node, containing information about the next router in the circuit. This process repeats until the circuit is fully established, with each router extending the circuit to the next node in the sequence.

As part of the circuit establishment process, each router assigns a unique Circuit ID to the circuit, allowing it to identify and process incoming data packets. The client and each router share symmetric encryption keys associated with the Circuit ID, enabling them to encrypt and decrypt data exchanged within the circuit. These encryption keys are used to secure the communication channels between the client and each router, ensuring the confidentiality and integrity of the transmitted data. This is illustrated with an example scenario as represented in Figure 2.5:

a. Negotiation and Key Exchange:

The client negotiates encrypted connections with each selected node, exchanging public keys (PK) and generating shared keys (KA, KB, KC) using asymmetric encryption techniques. The client and each node agree on symmetric encryption keys (KA, KB, KC) for secure communication within the circuit through the key exchange. The client initiates the circuit establishment process by selecting Guard Node A as the entry point.

b. CREATE and EXTEND Cells:

The client sends CREATE cells to Guard Node A, requesting the establishment of the circuit. Guard Node A processes the CREATE cell, selects Middle Node B as the next node, and sends an EXTEND cell to Middle Node B. Middle Node B receives the EXTEND cell, extends the circuit to Exit Node C, and forwards an EXTEND cell to Exit Node C.

c. Circuit ID and Encryption Keys:

Each node assigns a unique Circuit ID to the circuit and shares symmetric encryption keys (KA, KB, KC) with the client. These encryption keys are used to secure the communication channels between the client and each node, ensuring data transmission confidentiality and integrity.

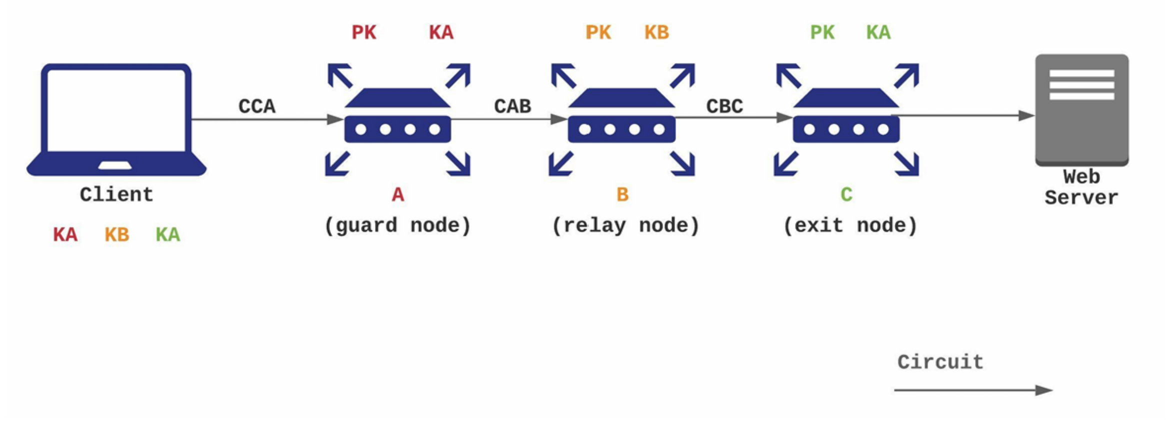


Figure 2.5: Circuit Establishment scenario with a client and three intermediary nodes.

2.3.1.3. Communication with Web Server

Once the circuit is established, depicted in the diagram provided, the client gains the capability to transmit requests through the onion network. To do so, the client crafts a relay cell with the “begin” command, containing details of the desired server (IP address) and port for connection. This message is encrypted with three layers of encryption using the shared keys KC, KB, and KA. As the relay cell traverses the circuit CCA, Router A, recognizing its origin from the specific client, applies its shared key KA to decrypt the first layer of encryption. Subsequently, the client forwards the encrypted cell along the established circuit.

Router A, unable to decipher the decrypted result, relays the cell to the next hop in the circuit connected via CAB. Prior to forwarding to Router B, Router A modifies the circID field to CAB. Upon receipt, Router B follows the same decryption process as Router A. Upon reaching the exit node (Router C), the node decrypts the cell using KC, identifying the relay command “begin” and initiates a TCP connection with the specified server mentioned in the payload.

Upon receiving a response from the server, the exit node generates a relay cell with the relay command “data” and encrypts it with KC. The cell is then passed to Router B, encrypted with KB, and subsequently to

Router A. Upon arrival at the client, all layers of encryption are decrypted, verifying the establishment of the connection. The client can now send additional relay "data" commands through the circuit to communicate with the web server while ensuring anonymity.

The multiple layers of encryption (in this example, three) in this technique liken it to an onion, with each layer adding a level of protection. This analogy is illustrated in the accompanying diagram, showcasing a message originating from a client ready to be "peeled" off by the circuit it is traversing.

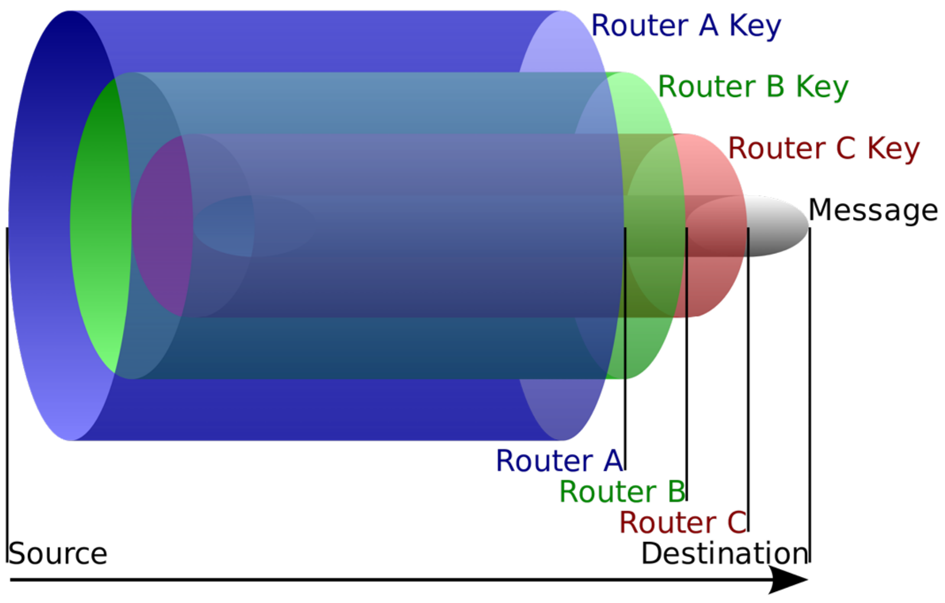


Figure 2.6: Multiple Layers of Encryption of a message by the client [13]

2.3.2. Onion Network

Now that we've explored the fundamentals of onion routing let's delve into its application in the Tor Project. When Tor is installed on our systems, it sets up a local host server called Onion Proxy (OP). This OP is an intermediary, managing the data flow between our applications and the Tor network. Upon initialization, OP fetches information about Tor's relays from directory servers and establishes a circuit with three nodes, as described in the onion routing concept.

Subsequently, we employ standard browsing mechanisms to interact with web servers: entering the desired address, DNS resolution, TCP handshake with the resolved IP, and initiating communication via HTTP or HTTPS. HTTPS ensures secure communication through SSL/TLS, which involves an additional exchange of information to establish an encrypted connection. However, these steps apply when accessing publicly visible web servers anonymously.

Traditional DNS mechanisms cannot be applied to access the Dark Web, where servers are hidden with onion services. Tor's onion routing can facilitate anonymous browsing across the Surface, Deep, or Dark Web. But, for the web servers to remain "hidden," they must operate within Tor's onion services. These services extend the onion routing concept, employing a three-hop circuit to conceal the client and server. The rendezvous point is the meeting point between the client and server within this circuit. The Tor Onion Service mechanism is clearly explained in 2.3.2.4

It's crucial to note that while Tor clients always benefit from a three-hop circuit for protection, web servers are safeguarded only when hosted within the onion services layer of the Dark Web.

2.3.2.1. Tor Nodes

Tor nodes are individual servers that volunteer to participate in the Tor network by running Tor software. These nodes receive, forward, and deliver encrypted user traffic across the network. Each Tor node operates independently and contributes to the overall functionality and resilience of the Tor network. Relays and bridges are the two nodes serving different purposes in the Tor network. According to the metrics of Tor, there are over 8000 relays and 2000 bridges in the Tor Network.

1. **Tor Relays**

   Relays are the primary nodes that make up the backbone of the Tor network. Relays receive encrypted traffic from one node and forward it to another node in the Tor circuit. There are three main types of relay nodes, each serving a distinct purpose in the routing process:

   a. Entry Nodes (Guard Nodes): Entry nodes act as the initial point of contact for user traffic entering the Tor network. When a user initiates a connection to the Tor network, the entry node receives the encrypted traffic and forwards it to the next node in the circuit. Entry nodes cannot access the traffic's final destination due to the layered encryption used in Tor (onion routing).

   b. Middle Nodes: Middle nodes relay encrypted traffic between the entry and exit nodes. Middle nodes do not know the origin or final destination of the data packets they relay, ensuring the traffic remains anonymous throughout its journey within the Tor network.

   c. Exit Nodes: Exit nodes are the final point in the Tor network where the final layer of encrypted traffic is decrypted and sent to its intended destination on the clearnet. As exit nodes handle unencrypted traffic, they pose a potential privacy risk if the traffic is not encrypted end-to-end.

2. **Tor Bridges**

   Bridges are special-purpose relays designed to help users bypass internet censorship and access the Tor network in regions where Tor is blocked or restricted. Bridges are not publicly listed in the main Tor directory to prevent them from being easily blocked by censors. Users can obtain bridge addresses from trusted sources or directly from the Tor Project's website to discreetly connect to the Tor network.

2.3.2.2. Onion Services

Onion services, a fundamental feature of the Tor network, were introduced in 2002. These services enable web servers to operate and interact with clients while maintaining anonymity. Unlike traditional web servers, which reveal their IP addresses when connecting with clients, onion services allow servers to accept connections and exchange data without disclosing their identities. This anonymity feature is particularly valuable for servers that store and provide sensitive content, where owners prioritize confidentiality and wish to remain unknown.

To grasp the concept of onion services, consider an analogy. Imagine both the client and server as participants in a secret rendezvous. In this scenario, Figure 2.9b, not only does the client's identity remain hidden through the Tor network, but the server also gains the ability to operate incognito. To achieve this anonymity, the server, like the client, operates behind an onion proxy, creating a circuit that enables them to rendezvous at a specified node known as a rendezvous point. This rendezvous point acts as a neutral meeting ground where the client and server can exchange data without directly communicating, thereby preserving their anonymity.

Understanding the addressing system used by Onion Services is crucial. Traditional web servers rely on DNS to map domain names to IP addresses, but onion services operate differently. Given the need for anonymity, these servers do not share their IP addresses openly. Instead, they utilize a unique addressing system within the Tor network, ensuring that the server's location remains concealed while allowing clients to find and connect to the desired hidden service. This specialized addressing mechanism ensures that Onion Services can provide content discreetly and securely, catering to the privacy needs of both server owners and clients.

2.3.2.3. Onion Address

Onion addresses, commonly known as .onion addresses, are special domain names used within the Tor network to identify hidden services. Unlike traditional domain names that rely on the Domain Name System (DNS) to map human-readable names to IP addresses, onion addresses function within the Tor network's decentralized and anonymous environment. These addresses allow users to access websites and services hosted on the Tor network anonymously and securely. A V3 onion address is a

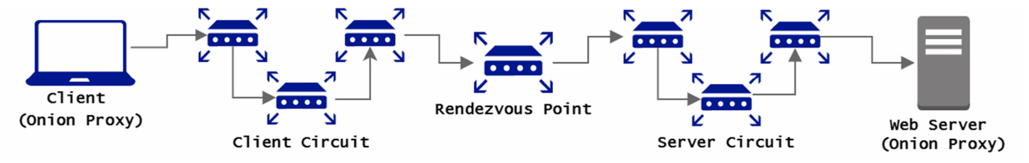


Figure 2.7: 6-hop circuit when a client communicates an onion service

56-character alphanumeric string ending with the .onion suffix. The structure of a V3 onion address is as follows: `ioiogw54kw4vqidvd274f53avjdj4jbw46ubwxvo3cjv7culwjnuiaqd.onion`

a. Key Generation: A V3 onion service generates a long-term Ed25519 key pair consisting of a private key (32 bytes) and a public key (32 bytes).

b. Address Generation: From the public key, a hash is derived using the SHA3-256 function. The resulting hash is truncated to produce the 56-character Base32 encoded string, forming the onion address.

c. Checksum Calculation: The last two characters of the address serve as a checksum. They are derived from the first few bytes of the SHA3-256 hash of the address.

The use of cryptographic operations to generate onion addresses has several advantages: It allows for easy verification of the authenticity and ownership of onion addresses. By matching the hash result of a server's public key with its onion address, users can be confident that they are connecting to the legitimate and intended server, reducing the risk of man-in-the-middle attacks or spoofing. It also enables a decentralized approach to address generation, eliminating the need for central authorities or DNS services.

2.3.2.4. Tor Onion Service Mechanism

In the context of the surface web, the interaction between a client and a server is more straightforward and transparent, as shown in Figure 2.8:

1. If the client is using a domain name, a Domain Name System (DNS) query is triggered. This process aims to map the domain name to the corresponding IP address.
2. The DNS server sends the IP address back to the client, completing the resolution process.
3. The client establishes a direct connection with the server using the server's IP address.
4. The server processes the client's request and returns the requested information or resource to the client.

This simplicity stands in stark contrast to the layered anonymity provided by the Tor onion service mechanism [14] shown in Figure 2.9a:

1. As the first step, OS contacts Tor Relays and requests them to act as Introduction Points (IPOs). The OS employs the Tor network, strategically connecting to three IPOs (by default) via a long-term three-hop Tor circuit. By doing so, the OS effectively conceals and safeguards its identity behind the layers of the Tor network, allowing access exclusively through these IPOs.
2. With the introduction points established, the next step is to create a mechanism for clients to discover them. The OS constructs an Onion Service descriptor(OSD) to achieve this. OSD is a cryptographic document that includes information about the service's IPOs, public key, and other relevant details. Once the descriptor is signed, the OS uploads it to a distributed hash table (DHT) integrated into the Tor network. This descriptor is then signed using the Onion Service's identity private key, the private component of the public key encoded in the Onion Service address. A Distributed Hash Table is a decentralized, distributed system that provides a lookup service similar to a hash table, allowing for the efficient storage and retrieval of key-value pairs across a network of nodes. In Tor, the DHT is used for specific purposes, primarily related to hidden services.
3. A Tor-enabled client attempting to connect to the OS retrieves the signed OSD from the DHT. Within this OSD are the details, such as IPOs, that enable clients to initiate communication with the OS.

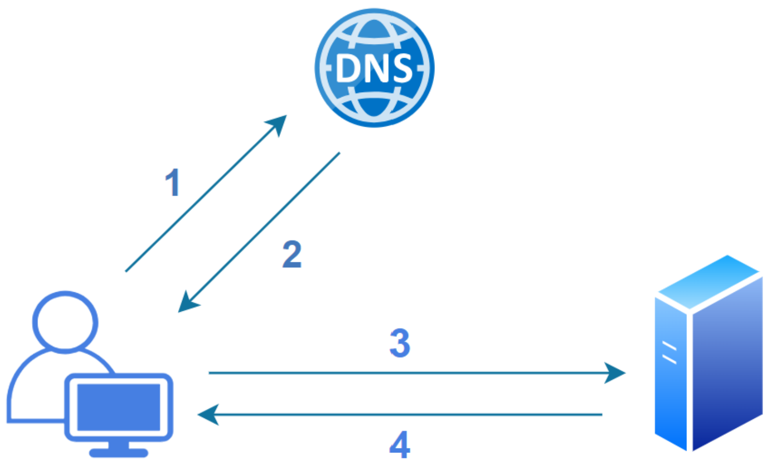


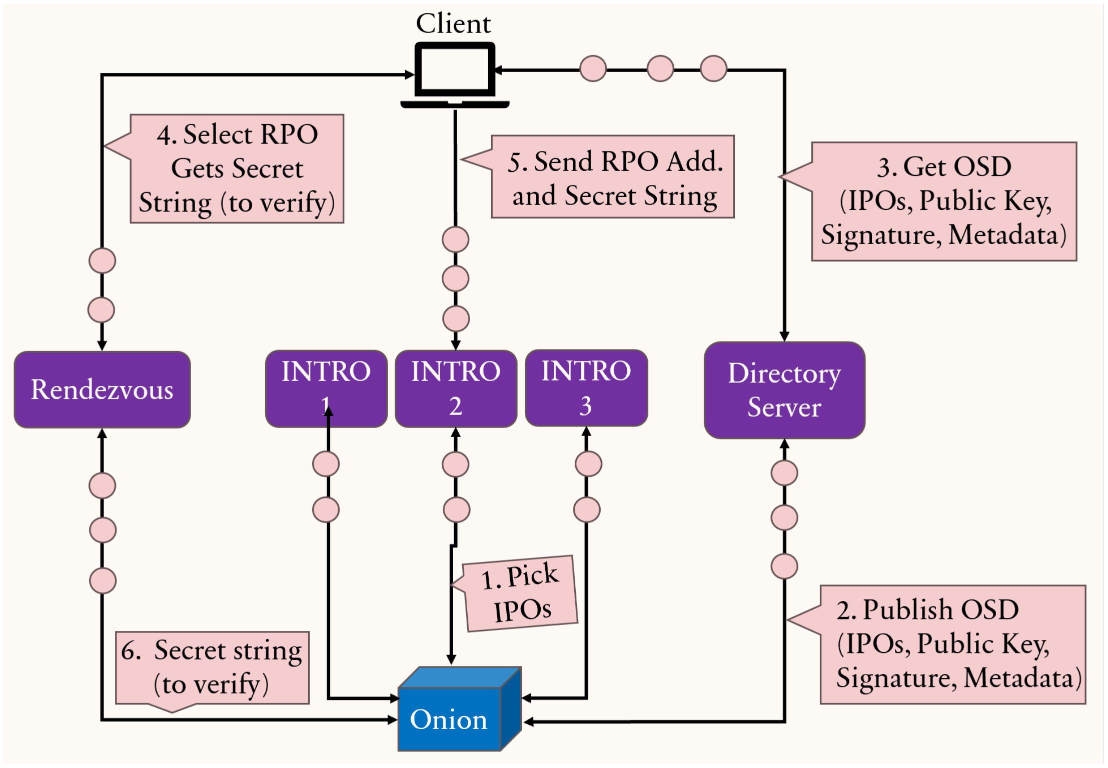
Figure 2.8: Accessing a service from Surface Web

4. Before the introduction, The Tor client randomly selects an intermediary relay node called Rendezvous Point (RPO). As part of the rendezvous procedure, the RPO provides the client with a unique "one-time secret." This secret is a cryptographic token that will be utilized in the rendezvous process to ensure the integrity and security of the communication between the client and the OS. Now, the client is ready to facilitate the secure introduction between the user and OS. The RPO plays a crucial role in managing the secure communication between the client and the OS without compromising the anonymity of either party.
5. Subsequently, the client decrypts the OSD to obtain IPO details and sends a request—comprising RPO information and a secret string to one of the IPOs. IPOs pass this information on to the OS.
6. The OS first verifies the authenticity of the details received from the IPO. Cryptographic techniques, such as signature verification, may be employed to ensure that the details provided by the client have not been tampered with during transmission. and connects to the RPO. If the client passes all verification checks, the OS connects to the RPO and sends a one-time secret. The RPO conducts a final verification process by matching the secret strings from the client and the service.
7. With the successful verification, the RPO transitions to a relaying role. It facilitates end-to-end encrypted communication between the client and the OS through a 6-hop circuit as shown in Figure 2.9b. Messages relayed by the RPO are encrypted, ensuring that the content remains confidential and secure during transmission.

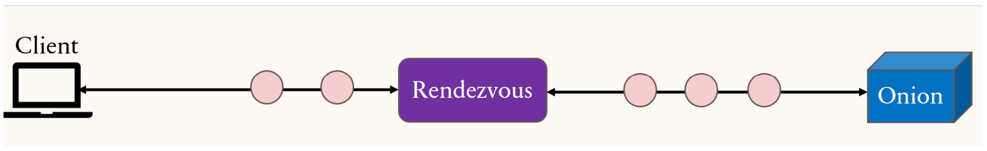
Onion proxies produce a new server descriptor and an updated extra-info document under the following circumstances: 1. When a predefined period (default is 18 hours) has elapsed since the last descriptor generation, 2. If any descriptor field, excluding bandwidth or uptime, undergoes a change, 3. If the server's uptime is less than 24 hours and the bandwidth has doubled since the last descriptor generation, provided that a specific time interval (default is 3 hours) has passed since the bandwidth change, 4. When the server's uptime is reset due to a restart, 5. Upon receiving a network status consensus where it is not listed, 6. Upon receiving a network status consensus, it is listed with the StaleDesc flag.

2.4. Port Scanning

This section outlines the distinctions between open, closed, and filtered ports and delves into the significance of port scanning, elucidating their roles in network security. Additionally, the section explores various port scanning methods and tools commonly employed in cybersecurity practices.


Client
4. Select RPO Gets Secret String (to verify)
5. Send RPO Add. and Secret String
3. Get OSD (IPOs, Public Key, Signature, Metadata)
Rendezvous
INTRO 1
INTRO 2
INTRO 3
Directory Server
1. Pick IPOs
2. Publish OSD (IPOs, Public Key, Signature, Metadata)
6. Secret string (to verify)
Onion

(a) Accessing a Tor Onion Service



(b) 6 hop circuit between the client and onion

Figure 2.9: Tor Onion Service Mechanism

2.4.1. Significance of Port Scanning

Port scanning involves systematically probing a target network or system to discover open ports and the associated services running on them. It sends network packets to specific port numbers and then analyses the responses to see if the ports are open, closed, or filtered. An open port refers to a network communication endpoint on a computer system that actively accepts incoming connections from other devices or services. When a port is open, it indicates that a service or application is running on the system that is actively listening for and responding to incoming network requests on that specific port. Open ports are accessible for communication, allowing data to be transmitted and received between the system and other devices or services over the network. In contrast, a closed port denotes a network port on a computer system that is not actively accepting incoming connections. When a port is closed, it means that no service or application is running on the system that is listening for incoming network requests on that particular port. Closed ports are typically protected by firewall configurations or network security measures to prevent unauthorized access or communication attempts. While closed ports do not respond to incoming connection attempts, they contribute to securing the system by limiting potential entry points for attackers. A filtered port refers to a network port on a system whose status is ambiguous, as it neither confirms nor denies incoming connection attempts. When a port is filtered, it suggests that the system is not actively

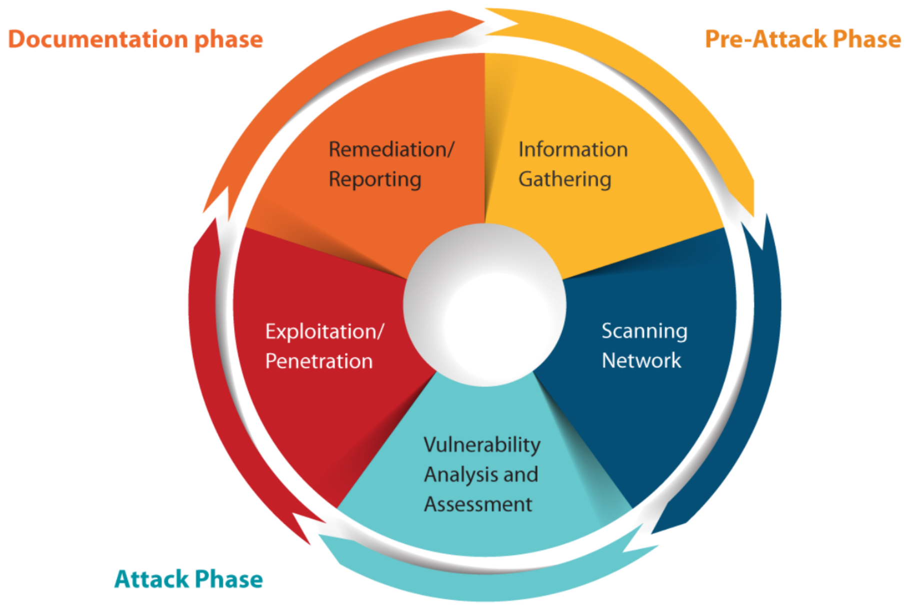


Figure 2.10: Methodology of Penetration Testing. Retrieved from [16]

responding to connection requests on that specific port. This lack of response can occur due to various reasons, including firewall rules, network congestion, or deliberate blocking by network devices. Filtered ports do not provide definitive information about the state of the port, but they indicate that communication attempts are being restricted or obstructed in some manner, requiring further investigation or configuration adjustments. The key difference between closed and filtered ports lies in the response behavior: closed ports actively respond to connection attempts with a closure notification, while filtered ports do not respond, leaving the status ambiguous.

Attackers can gain valuable information about potential entry points and vulnerabilities in the target system by identifying open ports. According to EC-Council Certified Ethical Hacker (CEH), penetration testing can be divided into five phases as shown in Figure 2.10. We can see that scanning is the second step in penetration testing. In this phase, the tester uses various port scanning and vulnerability scanning tools to identify open ports and other backdoors. Since open ports serve as potential ingress points for adversaries, detecting and cataloguing them is paramount for subsequent phases. While scanning can reveal potential threats, it falls short of determining the extent to which hackers may exploit vulnerabilities, necessitating human intervention to reach its full potential [15].

2.4.2. Port Scanning Methods and Tools

Some of the common port scanning methods are TCP SYN Scan, TCP CONNECT Scan, TCP ACK Scan, Stealth Scan (TCP NULL, FIN, and XMAS Scans), and UDP Scan [17].

TCP SYN scan is one of the most commonly used methods for port scanning. A TCP SYN packet is sent to determine the port status in this case. The port is open and listening if the response is a SYN/ACK. An RST packet means the port is closed. In addition, the port is considered filtered if there weren't any responses after multiple requests.

TCP Connect Scan, or vanilla scan, establishes a full TCP connection through a three-way handshake. This method, while comprehensive, is slower and requires more packets than a SYN scan. Furthermore, a huge number of TCP connection requests in a short period of time is suspicious, and it is more likely to be logged in the target system, thus increasing the likelihood of being detected by IDS/IPS systems.

TCP ACK scan differs from other scanning methods. In this case, the TCP probe packet contains only the ACK flag. A response of RST indicates an unfiltered port, but whether it's open or closed remains unknown. The main purpose of the ACK scan is not to determine the port state but to identify firewall rules.

With UDP Scan, the scanner sends a UDP packet to the target. This method can only detect closed ports due to the nature of the UDP protocol, which does not require a formal connection establishment. When a UDP scan is initiated, the scanning tool sends UDP packets to target ports and waits for responses. An ICMP port unreachable error signifies a closed port, while other ICMP unreachable errors indicate a filtered port. If the port is open, the target system typically does not respond with any acknowledgement, as UDP does not require it. Therefore, the scanner cannot reliably determine if the port is open based on a lack of response. Hence, UDP scanning less effective than TCP scanning in certain scenarios.

Stealth scanning, often executed through FIN, Xmas, and NULL scans, is a method used to probe target systems while attempting to evade detection by intrusion detection systems (IDS) and firewall filters.

- *FIN Scan:* In a FIN scan, the scanner sends a packet with only the FIN flag set. Normally, a TCP connection is terminated with a FIN packet. If the targeted port is open, it should ignore the FIN packet, and no response will be received. However, the system should respond with a RST (reset) packet if the port is closed. The absence of a response indicates that the port is open or filtered, making this method particularly stealthy.
- *Xmas Scan:* The Xmas scan is named for the sequence of flags it sets in the TCP packet: FIN, URG, and PSH. Like the FIN scan, the Xmas scan relies on the behaviour of the targeted system's response. The system should respond with a RST packet if the port is closed. An absence of response indicates that the port is open or filtered.
- *NULL Scan:* In a NULL scan, the scanner sends a packet with no TCP flags set, meaning the packet header is empty. According to TCP standards, this is an illegal combination of flags. The targeted port should respond with a RST packet if it is closed. The lack of response suggests that the port is open or filtered.

Some of the tools used for port scanning include Nmap, Zmap, Netcat, Masscan, and Unicornscan [18]. Table 2.1 gives an overview of the different port scanning tools and the different types of port scanning methods that the tools support.

Tool	SYN	CONNECT	ACK	UDP	FIN	XMAS	NULL
Nmap	✓	✓	✓	✓	✓	✓	✓
Zmap	✓	✗	✗	✓	✗	✗	✗
Masscan	✓	✗	✗	✗	✗	✗	✗
Netcat	✓	✗	✗	✓	✗	✗	✗
Unicornscan	✓	✓	✗	✓	✓	✓	✓

Table 2.1: Overview of port scanning tools and respective scanning methods

3 Related Work

This chapter reviews significant and contemporary studies within protocol analysis and diverse port-scanning methodologies in the dark and clear web. Additionally, we pinpoint the primary challenges and constraints associated with existing approaches, elucidating how our work tackles these issues. Concurrently, we underscore the distinctive aspects that set our work apart.

3.1. Port Scanning in Clear Web

This section delves into the related works on port scanning within the clear web, beginning with protocol analysis and examining various scanning tools and methodologies.

3.1.1. Protocol Analysis

Nmap offers a prioritized port list found in the nmap-services file. This file contains data indicating the likelihood of each port being active. It is based on an extensive Internet-wide port scan conducted in 2008, supplemented by data from various organizations about their internal networks. This list has undergone updates and adjustments since its start but has not seen major revisions [5].

By default, Nmap leverages this data to scan only the top 1,000 ports, a feature introduced in 2008. Before this enhancement, Nmap scanned all lower-numbered ports and specific higher-numbered ones. The shift to scanning the top 1,000 ports significantly improved Nmap's performance by increasing the number of open ports detected while reducing scan time. Additionally, Nmap allows users to specify any number of ports to scan and to limit scans to ports that meet a certain frequency threshold.

Based on the statistics of Nmap [19], the top 20 most common ports are 21 (FTP), 22 (SSH), 23 (Telnet), 25 (SMTP), 53 (DNS), 80 (HTTP), 110 (POP3), 111 (RPCBind), 135 (MSRPC), 139 (Netbios-SSN), 143 (IMAP), 443 (HTTPS), 445 (Microsoft-ds), 993 (IMAPS), 995 (POP3S), 1723 (PPTP), 3306 (MySql), 3389 (Ms-Wbt-Server), 5900 (VNC), 8080 (Http-Proxy).

3.1.2. Port Scanning Tools and Methodologies

A comparative study of port scanning techniques was proposed in [20] to evaluate their impact on the scanned hosts performance. Three scanning techniques were compared: TCP SYN, TCP Connect and UDP scan and several experiments were conducted using NMAP, Unicornscan, Netcat. Of the three port scanning techniques, TCP SYN scan has the least impact on the targeted scanned host.

The authors of [21] compared eight port scanning tools based on 15 evaluation criteria. The tools compared were Nmap 5.51, SuperScan 4.0, Advanced Port Scanner, Advanced Administrative Tools, Angry IP Scanner, Atelier Web Security Port Scanner, Unicornscan and GFILANguard. Some important criteria were based on the latest update released, scan range possible in single entry, ability to perform TCP SYN and UDP Scanning, banner grabbing feature, database integration in the tool and whether the tool is open source or not). Their evaluation declared that Nmap is the best scanning tool having the most robust features. Amongst all the scanning tools, The Advanced Scanner and AngryIP Scanner satisfied the fewest criteria.

Existing literature in the field of port scanning tools often focuses on comparing the performance and capabilities of different tools and categorizing various scan types and techniques. However, there is a notable gap in the literature regarding the efficacy of these tools, particularly in terms of false positives, true negatives, and resource utilization (CPU and RAM). In a study by [12], the authors compare the efficacy and performance of three prominent port scanning tools: Nmap, Zmap, and Masscan. The study reveals that while the overall performance of these tools in terms of accuracy is comparable (achieving 100 percent accuracy), there are significant differences in resource utilization. Masscan emerges as the top performer in terms of both runtime, completing scans in just 2 seconds, and efficiently utilising CPU and RAM resources. On the other hand, Nmap, while exhibiting relatively low runtime (27 seconds), consumes the highest amount of RAM among the three tools. Meanwhile, Zmap, despite its longer runtime of 644 seconds, demonstrates lower resource utilization than Nmap but higher than Masscan.

3.2. Port Scanning in Dark Web

Port scanning within the dark web, specifically through the Tor network, presents unique challenges and insights compared to the clear web. This section examines the protocols and tools used for port scanning in the dark web, starting with a detailed analysis of Tor (The Onion Router) domains.

3.2.1. Protocol Analysis of Tor Domains

Some references in the literature have already explored the protocols of onion services. In 2014, [22] presented their findings on a content and popularity analysis of the 39,824 onion services. Among them, 24,511 remained accessible during the conducted port scans. The analysis revealed 22,007 open ports, with a significant majority of onion services identified as part of the Skynet botnet, notably open on port 55080. This particular port was found open on over 50% of all onion addresses, providing insights into the potential scale of computers infected by “Skynet.” Furthermore, HTTP and HTTPS services constituted 22%, while SSH services were operated by 5% of the onion services. According to [22], HTTP, HTTPS and SSH-based services were the most frequently detected protocols in the network. However, the absence of SMTP and Bitcoin protocols suggested a change in service usage between 2014 and 2019. This finding is further confirmed in a more recent study [23] and [24], which shows that anonymous email servers and Bitcoin clients have become significantly more popular in recent years.

The statistics in 2018 [23] show that out of 14,972 onion services, 54.6% were HTTP, 26.1% Zeronet (15441), 12.1% SSH, followed by 2% HTTPS, 1% of Mail, IRC and Ricochet and 0.3% Bitcoin. A study [24] carried out in 2019 found that out of 60,036 onion services, 1370 unique ports were detcted and belonged to 219 unique protocols. HTTP was the most commonly used protocol with 82%, followed by 7.7% SSH and 4.2%SMTP. Most HTTP services were found on port 80, while SSH services were most commonly on port 22 and SMTP services on port 25. Only a few onion services, 1.8%, use HTTPS and 1.3% related to bitcoin.

All the mentioned research, however, is confined to Tor v2 onion services. The state-of-the-art paper by [25] explores the availability and protocols of Tor v3 domains, performing a port scan on 41,583 onion services for 15 ports. The findings revealed HTTP as the most used protocol with 93%, followed by SSH and HTTPS with 1.1% and 0.62%, respectively. Cryptocurrencies such as Monero and Bitcoin were found to mainly use ports 18081 and 8333, constituting 0.08%. Less commonly used protocols included instant messaging services (Jabber/XMPP and IRC) at 0.05%, IMAP at 0.03%, and FTP at only 0.004%.

In summary, the analysis of the protocols used by onion services within the Tor network has revealed that HTTP is the most commonly used protocol, followed by SSH. This study aims to revisit the distribution of active protocols, particularly for a sample of version 3. The analysis will also expand the list of protocols that have not been considered before to explore the potential use of other protocols in the dark web.

3.2.2. Port Scanning Tools and Methodologies

Moving to port scanning tools and methodologies, while most literature focuses on protocol analysis, only [24] provides insights into their port scanning methodology. They conducted test runs with common port scanners such as Nmap, Ncat, and Netcat. Their findings indicated that Ncat performed the fastest for single-service scans, while Nmap outperformed all other scanners for scans involving a set of addresses. An incremental scan approach was adopted, where only an increasing subset of ports was scanned until at least one open port was detected. The scanning started with commonly used ports like HTTP 80 and HTTPS 443. Subsequently, a variety of common Tor applications, such as TorChat, Jabber, common IRC and RPC ports, Torrent tracker and ports known to be used by the Skynet and Trickbot botnets, were

Year	Author	Tor Version	Protocols	Number of Onions
2014	Alex Biryukov et al. [22]	Tor v2	HTTP/HTTPS: 22%, SSH: 5%, Skynet: 50%	39,824 onions, 24,511 accessible 22,007 open ports
2018	Gareth Owenson et al. [23]	Tor v2	HTTP: 54.6%, Zeronet: 26.1%, SSH: 12.1%, HTTPS: 2%, Mail/IRC/Ricochet: 1%, Bitcoin: 0.3%	14,972 onions
2019	Martin Steinebach et al. [24]	Tor v2	HTTP: 82%, SSH: 7.7%, SMTP: 4.2%, HTTPS: 1.8%, Bitcoin: 1.3%	60,036 onions, 1370 unique ports 219 unique protocols
2019	Alejandro Buitrago López et al. [26]	Tor v3‘	HTTP: 93%, SSH: 1.1%, HTTPS: 0.62%, Monero/Bitcoin: 0.08%, Jabber/XMPP/IRC: 0.05%, IMAP: 0.03%, FTP: 0.004%	41,583 onions, 15 ports

Table 3.1: Summary of Related Works on Protocol Analysis of Tor Domains

examined. This is followed by an additional scan of the 100 most commonly used ports (as determined by Nmap’s list of well known ports [12]). Finally, if no open port is found, the scan continues with the 1,000 most common ports.

However, [24] did not provide time statistics for their scans. In contrast, [23] reported a scanning duration of 24 hours for 10,000 ports when investigating a single onion service. They employed a strategy focusing on the most probable open ports to optimise efficiency. Their approach involved focusing on the most probable open ports, informed by prior knowledge, gathering data from ongoing scans in the darknet, and conducting comprehensive scans on a selected subset of onion services. Ultimately, this comprehensive process resulted in a refined list of 40 ports.

4 Dataset

This thesis uses two primary datasets: the proprietary dataset obtained from CFLW Cyber Strategies and the Internet Assigned Numbers Authority (IANA) port mapping dataset. The following chapter highlights the characteristics and origin of the datasets, elucidating their significance in the context of this research.

4.1. Dark Web Monitor (DWM)

The Tor crawler, initially conceived as a research endeavour back in 2013, aimed to expand the scope of indexable content within the Dark Web beyond the limited set of seed Tor domains accessible on the clearnet. Over time, the wealth of data amassed by this crawler has evolved into a commercial offering by CFLW Cyber Strategies, known as the Dark Web Monitor (DWM). This proprietary tool, operating as an open-source intelligence (OSINT) solution, stands as the cornerstone of CFLW's intelligence services, meticulously crafted to furnish invaluable insights into the realm of criminality and fraudulence proliferating across the Dark Web. Utilizing state-of-the-art analytics, the Dark Web Monitor aids law enforcement agencies, cybersecurity firms, and financial institutions, empowering them to pinpoint suspicious activities and precisely investigate illicit infrastructures.

4.1.1. DWM Data Collection Method

DWM offers access to an extensive dataset compiled from the Dark Web, meticulously indexed and made searchable. This dataset encompasses a wide array of domains, many of which host multiple pages that the crawler has systematically downloaded. During each download attempt, the crawler verifies the server's provision of HTML content, generating a snapshot by storing the HTML file. Subsequently, DWM conducts a comparative analysis of current and previous downloads to identify any alterations. In the event of detected changes, a new version is generated. This functionality facilitates users in monitoring domain modifications and utilizes DWM akin to a wayback machine for investigating domain histories.

The Tor crawler undergoes continuous refinement to enhance its capabilities and efficacy. The crawler maintains a dynamic list of Tor onion domains, continually adding new entries as they are discovered during the crawling process. Each onion domain is systematically crawled regularly, ensuring comprehensive coverage and indexing of active and transient domains. Utilizing a "snowballing" approach, the crawler traverses the interconnected pages within each domain, recursively uncovering new entries and expanding its dataset.

When a Tor domain experiences downtime or becomes inactive, the crawler implements a strategic re-visit schedule with a recurring interval of one hour. If the domain remains offline after three consecutive attempts, the crawler adjusts its crawling schedule using an exponential back-off approach. This entails revisiting the Tor domain at intervals of 18, 36, and 72 hours, respectively, with the maximum revisit frequency capped at 10 days. Employing exponential back-off mechanisms, the crawler adapts its crawling frequency based on the availability and responsiveness of each domain, ensuring optimal resource utilization and data integrity.

DWM employs advanced content analysis techniques, leveraging regular expressions to extract critical information such as cryptocurrency addresses, PGP keys, and email addresses from crawled pages. This extracted data is meticulously archived in cloud storage repositories, facilitating further analysis and investigation. Furthermore, each discovered domain undergoes classification and receives corresponding tags. These tags encompass various abuse types (such as financial crime, sexual abuse, violence, etc.) and service categories (like shops, file sharing, service providers, etc.), providing users with comprehensive insights into the nature and purpose of each domain.

4.1.2. DWM Dataset

The dataset utilized in this research project is a proprietary collection obtained from CFLW Cyber Strategies, with access granted through their Dark Web Monitor. This monitoring tool actively crawls onion services, maintaining a comprehensive record of their status, historical changes, and related information. The dataset encompasses a substantial volume of 0.5 million onion services, making it a notable resource for comprehensive analysis. The status of these onions detected by the DWM is solely web-based, i.e., it checks the status on ports 80/443, but there might be ports other than 80/443 open. This dataset helps me bootstrap the process of port scanning.

One critical aspect of the dataset is that the status of these onion services detected by the DWM is solely web-based, i.e., it checks the status on ports 80/443. However, there might be other ports open on these onion services that are not captured by this initial monitoring. By providing a substantial base of onion services with known statuses on standard web ports, the dataset significantly helped bootstrap the port scanning process and allowed targeted port scanning on a larger scale. This initial information streamlined the port scanning process, enabling the identification of additional open ports that may not have been immediately apparent through web-based monitoring alone. Crucially, this dataset facilitated a transition from a web-based to an application-based view. While the initial monitoring focused on identifying web services through ports 80/443, the port scanning process expanded the analysis to include any application running on these onion services.

Data collection spans September 2018 to January 2024, providing a longitudinal view of the evolving landscape of onion services on the Dark Web. The dataset is structured in JSON format, ensuring flexibility and ease of integration. The Dark Web Monitor offers an API that enables automated interactions with its data repository, providing users with a seamless way to access and analyze dark web-related information. The API is structured according to the JSON:API specification, making it compatible with various programming languages and platforms. This standardized format ensures interoperability and ease of integration with existing systems. These APIs enable users to: 1. Retrieve a comprehensive list of darknet domains, 2. Access detailed information pertaining to individual darknet domains, 3. Obtain a list of pages associated with a specific domain, and 4. Retrieve a comprehensive list of crypto-assets, PGP keys, and email addresses linked to the current domain.

The API focused on providing a list of darknet domains is a pivotal component of this research endeavor. It encompasses the following key fields essential for our investigation as represented in Table 4.1

The darknet domains API offers users a wide range of filtering options to obtain a list of darknet domains relevant to their needs. Additionally, all columns in the Darknet Domains list view support sorting in ascending and descending order, allowing users to organize domains based on their preferred criteria. For example, users can prioritize displaying domains with the highest number of tags or those with the most pages within the filtered domains. Some of the available filter options are represented in Table 4.2

This dataset is particularly relevant to the research on port scanning of onion services. It provides essential information about onion service status, historical changes, and associated details, forming the basis for an in-depth exploration of port configurations and services running on the Dark Web.

4.2. IANA Port Mapping with Services

The IANA Port Mapping Dataset is a fundamental resource in this study, offering a comprehensive mapping of port numbers to their corresponding services as defined by the IANA [28]. Originating from the authoritative registry responsible for managing port assignments for various protocols, this dataset provides essential information for understanding the functionalities associated with different network ports. By leveraging the mapping between port numbers and services provided by the dataset, open ports identified in dark web datasets were grouped and categorized based on their associated services. This integration enabled a more granular understanding of the services accessed or targeted within the dark web environment, shedding light on potential security threats and vulnerabilities.

Column	Description
DomainID	A unique identifier is assigned to each Darknet domain.
Title	The title of the home page associated with each onion service. This field provides insights into the content or purpose of the service, aiding in the contextual understanding of the service's potential offerings.
DiscoveredAt	Timestamp indicating when the onion service was initially discovered by DWM. This temporal data is pivotal for conducting longitudinal analyses and tracking changes in the Dark Web landscape over time.
Status	Indicates whether an onion service is currently online or offline when last checked by the DWM's crawler (approximately every 18 hours).
Uptime	Represents the duration observed by the DWM for which the onion service has been continuously available. Uptime is a critical metric for assessing the reliability and stability of services, offering insights into their operational characteristics.
Pages	The count of pages within the subdirectories originating from the root directory of a domain that has been successfully traversed and indexed by the DWM's crawler.
Tags	A tally of the tags attributed to the domains. Hovering over the count will reveal the comprehensive list of tags linked with the respective domain.
Dupl	The count of mirror sites or domains with the same title
DN In	The count of Darknet domains in which the specified domain is referenced or found
DN Out	The count of Darknet Domains linked to or mentioned on the domain's pages

Table 4.1: Key Fields of Darknet Domain API [27]

Filter	Description
Domain search	Filter domains based on a specific string of characters in their URL.
Title search	Filter domains based on a specific string of characters in their title.
Title unique	Filters out domains with duplicate titles, displaying only one domain per title.
Status - Online / Offline	Filters domains based on their online and/or offline status.
Type	Filters domains based on a specific darknet, such as Tor v3, I2P, or others
Start Date	Filters domains discovered after a specific date and time.
End Date	Filters domains discovered before a specific date and time.

Table 4.2: Filter Options of Darknet Domain API

5 Methodology

This chapter outlines our approach to investigating the Tor onion services landscape. Firstly, we emphasize the importance of developing a robust port scanning tool. Secondly, we detail the core design principles guiding the tool's development. Next, we provide an overview of the system architecture, highlighting its key components and interactions. Following this, we explain our method for assessing the availability of onion services. Finally, we validate the efficacy and reliability of our port scanning tool, underlining its robustness.

5.1. Need for building Port Scanning Tool

The dark web presents unique challenges for port scanning due to its encrypted environment and the need for stealthy and efficient scanning techniques. While existing tools like Nmap, Ncat, and Netcat have been used for dark web port scanning, their slow performance and limited features make them impractical for comprehensive scans. We have seen in the literature that the time taken to perform a scan on 10000 ports for a single onion service took 24 hours. Previous studies, such as [21], have highlighted the limitations of existing port scanning tools in the dark web context. These studies have underscored the importance of optimizing scanning strategies and enhancing tool capabilities to achieve more efficient and comprehensive results. While some methodologies have been proposed, such as the incremental scan approach, time statistics for scans have often been lacking, leaving room for improvement in efficiency and accuracy.

5.2. Design Principles

In developing a specialized port scanning tool for the dark web, it is crucial to incorporate robust design principles that address this environment's unique challenges and demands. The Tor network's encrypted layers, the need for rapid and accurate scans, and the limitations of existing tools necessitate a thoughtful approach to tool design. By focusing on efficiency, accuracy, flexibility and scalability, we aim to create a port scanning tool that not only overcomes these challenges but also sets a new standard for dark web investigations. These design principles are the foundation for building a tool that can reliably identify open ports and adapt to evolving needs. The design principles guiding the development of the port scanning tool for the dark web include:

- **Efficiency:** The tool must be optimized using concurrent scanning techniques. Concurrency is a powerful design principle that can significantly increase the speed of port scanning by allowing multiple tasks to be executed simultaneously. In the context of the Onionscan port scanning tool, incorporating concurrency can optimize the scanning process, making it faster and more efficient. This efficiency is crucial for dark web investigations, where timely data collection is essential.
- **Accuracy:** Emphasis is placed on ensuring accurate and reliable scan results with comprehensive scanning capabilities. In sensitive environments like the dark web, accuracy is also a legal and ethical responsibility. Incorrectly identifying services could lead to legal ramifications or ethical concerns, especially when dealing with privacy-sensitive data.
- **Flexibility:** Users must be able to customize scanning parameters to suit specific requirements. This includes defining port ranges, specifying target onion services, and adjusting scanning intervals.

Such customization options ensure the tool can be tailored to different scanning scenarios, whether for broad sweeps of dark web services or focused investigations on particular targets.

- **Scalability:** The tool's support for concurrent execution ensures that it can scale efficiently as the size and complexity of the scanning tasks increase. This scalability is essential for handling large datasets or extensive dark web investigations, allowing the tool to perform effectively under high-demand conditions.
- **Data Storage and Analysis:** Integrating MySQL for systematically storing scan results is another aspect of the design principle. This database-driven approach organizes and preserves scanning data and lays the groundwork for advanced data analysis and visualization. Using tools like Python notebooks for data analysis, users can extend the tool's capabilities to perform in-depth investigations and derive actionable insights from the collected data.

5.3. System Architecture

This section presents a comprehensive overview of the system design employed for port-scanning onion services. The architectural framework is delineated into four key steps, each having a pivotal role in systematically exploring and analysing these services. Beginning with the Data Source, where information is gathered from the proprietary Dark Web Monitor, the subsequent step delves into the Port Scanning Module, elucidating the techniques and procedures employed in actively exploring onion services. Data Storage details the systematic organization and storage of the scanned data, laying the foundation for subsequent analysis. Finally, the Post Data Analysis step examines the obtained results, aiming to uncover patterns, anomalies, and trends within the port-scanning data. The synergy of these four steps forms a robust and structured approach to comprehensively understand and analyze the landscape of onion services on the Dark Web. The architectural flow is visually represented in Figure 5.1, providing a holistic view of the sequential processes involved in this research endeavour.

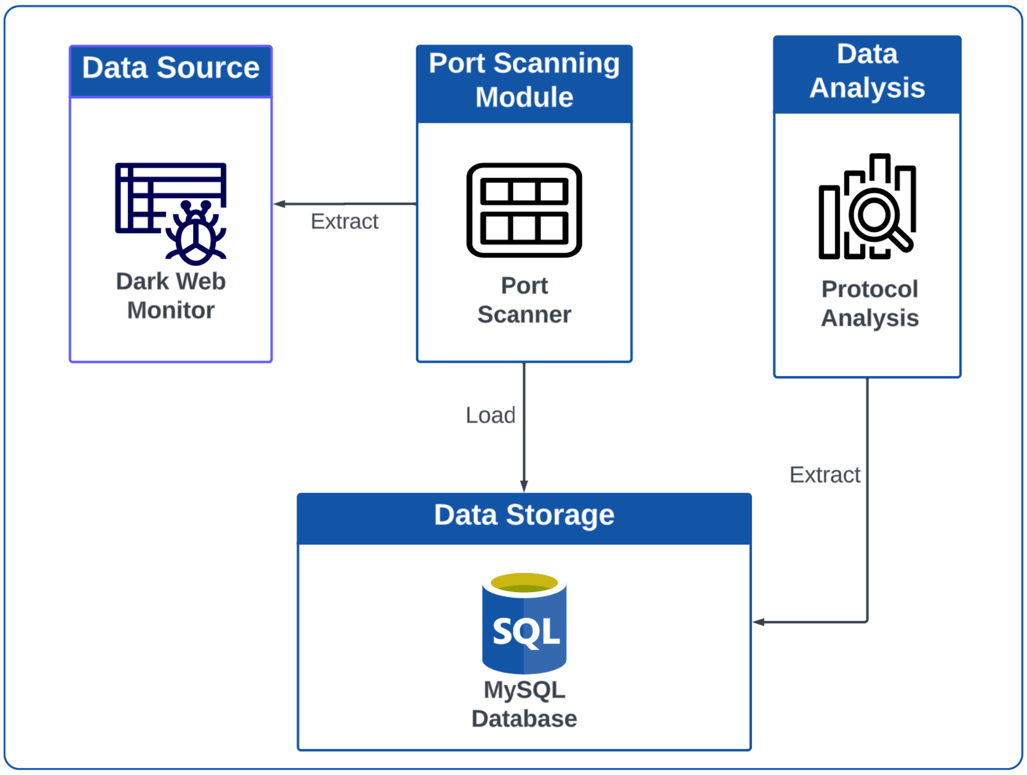


Figure 5.1: Framework Architecture

Data Source - The system begins by interfacing with the Dark Web Monitor API to fetch a curated list of onion services. This API is the primary source for obtaining up-to-date information on onion services.

Port Scanning Module - The core of the architecture involves a sophisticated port scanning engine tailored for onion services. The retrieved data is pre-processed to extract relevant information, such as onion service address and temporal metrics. This step ensures that the input to the port scanner is well-structured and conducive to effective analysis. The Port Scanning engine takes the following parameters as input: Tor Proxy Address, Port Range (with a default scan of the first 1000 ports in the absence of specification), and a file containing a list of onion services to be scanned (each line of the file has onion ID and onion address separated by a comma). The port scanning module leverages parallel processing techniques implemented through the Go programming language to enhance speed and efficiency. This optimizes the scanning process by simultaneously handling multiple onion services and ports, enabling the system to scale effectively. This scalability is crucial for accommodating the substantial volume of onion services within the dataset, ensuring a robust and responsive port scanning capability. In Section 5.4, an in-depth elucidation of the adopted port scanning approach is provided.

Data Storage - The outcomes of the port scanning analysis, including details on open ports detected and other metadata, are stored in a MySQL database. This relational database structure facilitates the secure and organized storage of results. The database consists of two tables, 'Scan Results' and 'Scan History.' The 'Scan Results' table employs 'OnionID' as the primary key, uniquely identifying each record. This table captures essential attributes such as the onion service's URL, discovery timestamp ('DiscoveredAt'), scan timestamp ('ScannedAt'), a list of open ports, scan status, last scan timestamp ('LastScannedAt'), and the specified port range. Concurrently, the 'Scan History' table features an 'ID' as its primary key, establishing a relationship with 'OnionID' as a foreign key, linking to the 'Scan Results' table. This table maintains a historical record of scans. The dual-table structure enables a comprehensive analysis of real-time and historical data, facilitating a detailed exploration of onion service port configurations over distinct timeframes.

Post Data Analysis - The post-analysis phase of this research encompasses a multifaceted approach to extract meaningful insights from the port-scanning data obtained from Tor v3 onion services. Utilizing statistical methods, the study aims to identify the most prevalent services within these domains, shedding light on the frequently encountered protocols. Correlation analysis will investigate the relationships between different ports, revealing potential dependencies and vulnerabilities associated with specific configurations. Adding a temporal dimension, the analysis will unfold patterns in the distribution of ports over time, providing a dynamic understanding of port usage trends. Additionally, examining status changes (Offline, Online, T.U) over time for each onion service will offer insights into the availability dynamics. Through visual aids like correlation matrices, temporal distribution charts, and status change charts, the post-analysis seeks to present these findings comprehensively. The outcomes of this phase are expected to contribute nuanced insights into commonly used services, an in-depth understanding of onion availability, and heuristics for optimizing future port-scanning strategies, thereby enhancing the overall understanding of Tor v3 onion services.

5.4. Approach

This section provides an overview of the tool's current port scanning methodology. It acknowledges the preliminary and basic nature of the approach, setting the stage for a deeper exploration and refinement as the thesis progresses and more data is collected. As we delve further into our analysis, we aim to develop heuristics that will enhance and optimize future iterations of the port scanning strategy.

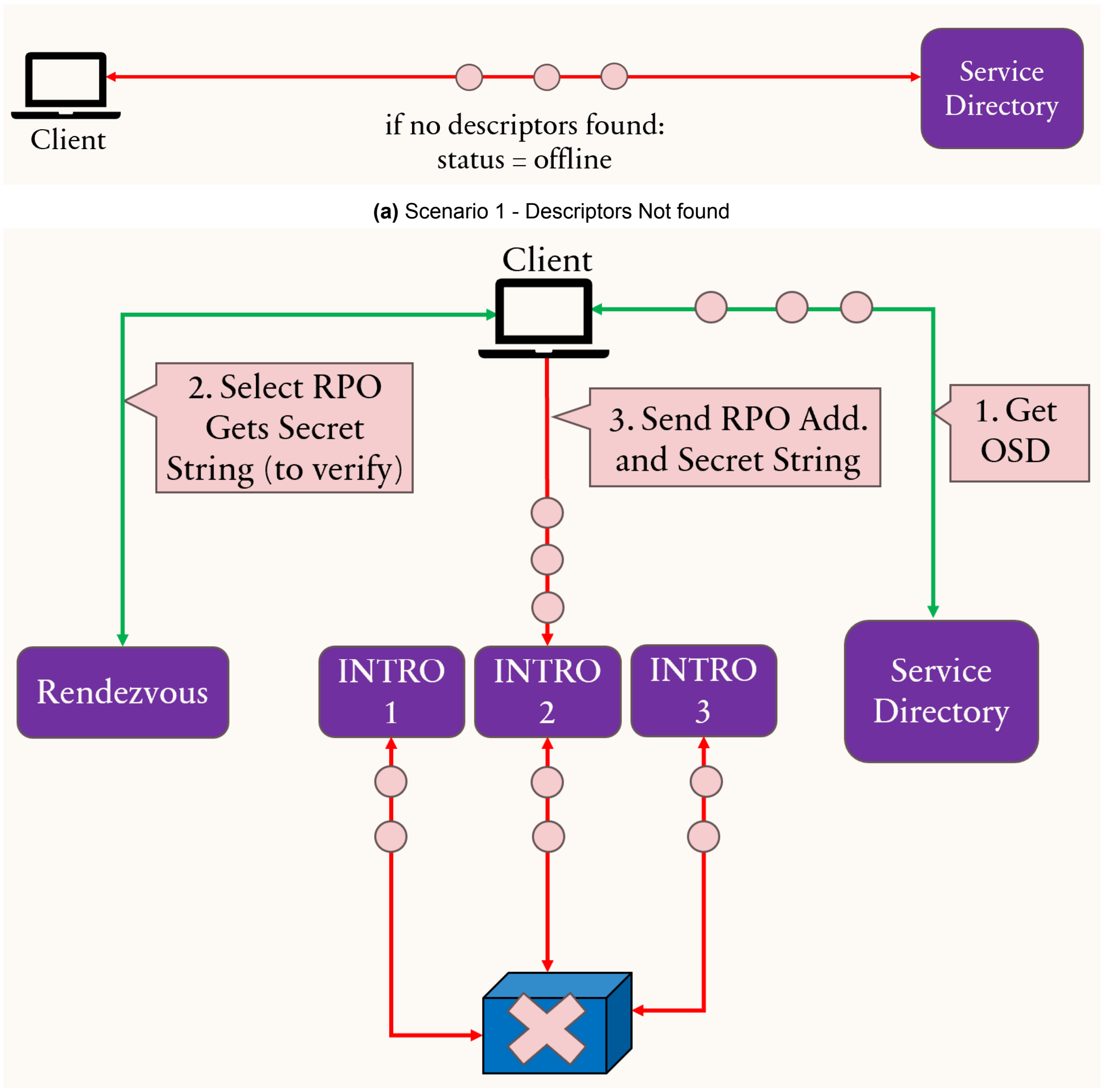
Before initiating the port scanning process for onion services, our primary objective is to ascertain the availability of the onions. The availability status of onion services is categorized into three distinct classifications: online, offline, or T.U. These classifications are elaborated upon in the 'Availability of Onion Services' section. Upon determining the availability status, our focus shifts to identifying the open ports if the onion is online. This sequential approach ensures a structured and systematic examination of onion services, laying the groundwork for comprehensively evaluating their security posture.

5.4.1. Availability of Onion Services

Understanding the various status codes of onion services is crucial for comprehending their availability within the Tor network. They serve as indicators of the onion service's responsiveness and accessibility.

This subsection delves into the methodology and significance of classifying onion services using the three distinct status codes: Online, Offline, and T.U. Traditionally, the Dark Web Monitor (DWM) sends HTTP/HTTPS requests specifically to ports 80/443 and awaits responses from onion services. If a response is received, the service is categorized as ’Online’; if no response is received, it is labelled as ’Offline’. However, this approach may lead to inaccuracies. There are instances where ports other than 80/443 may be open, indicating that the onion service is online. Yet, it is incorrectly classified as ’Offline’ by the DWM due to its limited port focus. To address this limitation and refine our classification criteria, we transition from the conventional definitions of ’Offline’ and ’Online’, based solely on connectivity to ports 80/443, to more advanced definitions that are independent of the specific port accessed in the HTTP request. These refined status codes stem from a meticulous analysis comprising diverse experiments tailored to capture unique responses from onion services. This enhanced methodology offers a more nuanced understanding of onion service status, thereby boosting the accuracy and reliability of our assessments.

5.4.1.1. Offline



(a) Scenario 1 - Descriptors Not found

(b) Scenario 2 - Introduction Failed

Figure 5.2: Offline Scenarios

There are two scenarios in which an onion service can be considered offline. When a client initiates a connection request with an onion service, the initial step involves retrieving the onion service descriptors from the distributed hash table. In the first scenario, if an onion service remains inactive for an extended period or voluntarily de-registers itself from the Tor network, its corresponding onion service descriptors are purged from the distributed hash table. Consequently, when a client attempts to retrieve these descriptors, the Tor proxy generates an error indicating the unavailability of HSDirs and the absence of descriptors as shown in Figure 5.2a. Subsequently, the client receives a “Host unreachable” error response when attempting to establish a connection using the SOCKS5 proxy.

In Scenario 2, the process of fetching onion service descriptors and selecting a rendezvous point proceeds without any issues. However, the client encounters a failure during the introduction procedure. This failure can occur for two reasons: either all the introduction points are unavailable, or the introduction points are operational, but the onion service itself is offline. In some cases, despite an onion service not properly registering with the Tor network, its descriptors may still persist in the Distributed Hash Table (DHT). During this transitional phase, the initial step of fetching descriptors succeeds, but the subsequent introduction fails. Consequently, an onion service may be considered offline even when the introduction procedure encounters an issue. The Tor proxy reports that the introduction has failed, while the client receives a “host unreachable” response, indicating the inability to establish a connection with the onion service.

“An onion service is considered offline when it is not reachable within the Tor network either due to the descriptor unavailability or due to the failure of introduction phase” In other words, an onion service is said to be offline when any of the following occurs:

- Descriptor Unavailability: The service is considered offline when the Tor client fails to retrieve descriptors, preventing the establishment of connection information.
- Introduction Phase Failure: Unsuccessful attempts by the Tor client to connect to introduction points specified in the descriptors or when these introduction points cannot reach the onion service.

5.4.1.2. T.U

In this particular scenario, when a client initiates a connection to an onion service, the initial steps of fetching descriptors, finding a rendezvous point, and initiating the introduction procedure proceed successfully. However, as indicated by the red arrow in Figure 5.3, the fourth step encounters a failure. This failure can stem from various sources, such as issues within the Tor network, failure to establish a rendezvous point or temporary downtime of the domain’s server, potentially caused by an inability to handle the incoming traffic load. When a circuit is disrupted or the client receives no response due to the aforementioned reasons, the client’s Tor proxy enters a waiting state for a predefined duration. Subsequently, it encounters a connection timeout error or a Time To Live (TTL) expired error. To address these challenges, the status of an onion can be assessed by retrying the connection request a specified number of times (e.g., 3 times). This approach helps mitigate issues related to the Tor network or circuit connectivity problems.

“An onion service is considered to be T.U when a connection timeout response is elicited during the client’s attempt to establish a connection with the service after the Introduction phase.” In other words, an onion service is said to be T.U when any of the following occurs:

- Connection Timeout Response: A connection timeout response is encountered when the client’s attempt to establish a connection with the onion service exceeds the predetermined time limit for a successful connection.
- Possible Causes for Connection Timeout:
  - Rendezvous Point Unavailability: If the chosen rendezvous point becomes unavailable or experiences issues, the communication between the client and the onion service may be compromised, leading to TTL expiration.
  - Server-Side Congestion or Unavailability: The onion service’s server may experience congestion, high load, or temporary unavailability, causing the TTL to expire as the connection cannot be sustained within the expected time frame.
  - Resource Exhaustion on Tor: If the Tor network experiences resource exhaustion, such as running out of available circuits or processing capacity, it can contribute to TTL expiration.

5.4.1.3. Online

After successfully completing the initial steps of fetching descriptors, selecting a rendezvous point, and executing the introduction and rendezvous procedures, a 6-hop circuit is established between the client and the onion service. This circuit comprises three hops chosen by the client and three hops chosen by the onion service, with the rendezvous point situated as the third hop from the client’s side. Subsequently, the

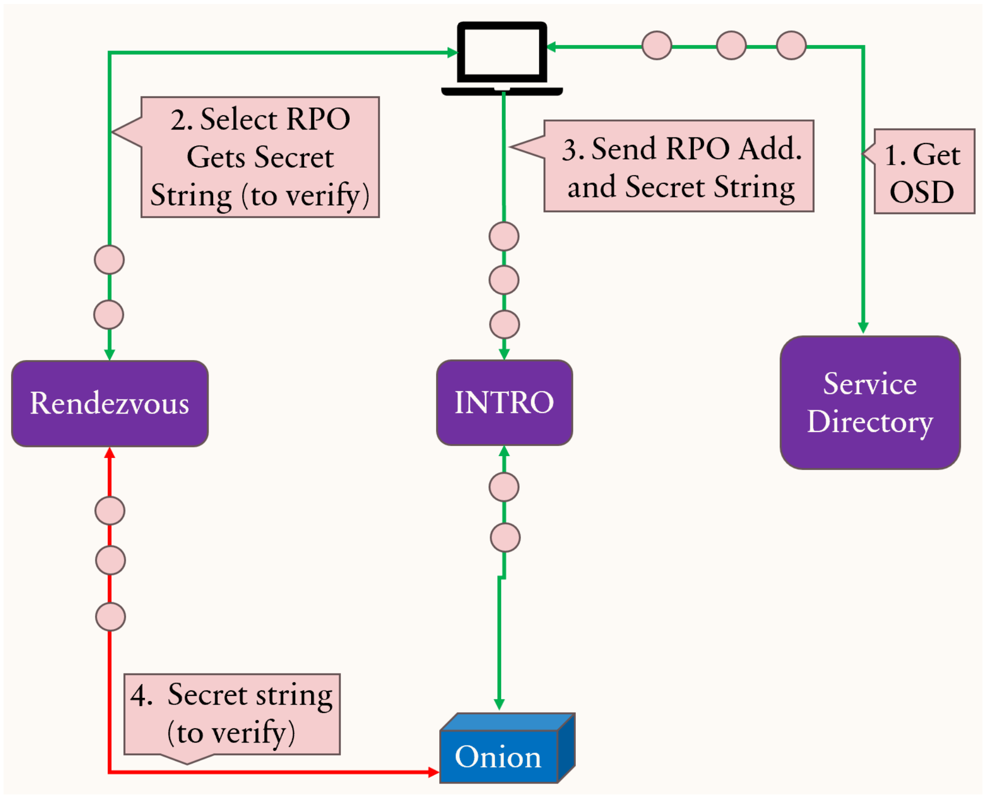

2. Select RPO Gets Secret String (to verify)
3. Send RPO Add. and Secret String
1. Get OSD
Rendezvous
INTRO
Service Directory
4. Secret string (to verify)
Onion

Figure 5.3: T.U Scenario

client forwards the connection request along the established circuit to the onion service. Upon receiving the request, the onion service may respond in one of two ways as shown in Figure 5.4:

1. Successful Connection Establishment: In this scenario, the client receives a response indicating successful connection establishment. No error is encountered, and the connection is seamlessly forwarded to the client. This outcome signifies that the port is open, allowing for uninterrupted connection establishment without any errors.

2. Error Response Received: Conversely, the client receives a "connection refused" error response from the onion service. A "connection refused" error from an onion service typically indicates that the targeted service is actively rejecting incoming connection attempts on the specified port. This refusal can stem from various underlying reasons. Firstly, the onion service might not be running or available on the designated port due to temporary downtime, misconfiguration, or intentional blocking. Secondly, firewall restrictions or network configurations could be in place, rejecting incoming connections to protect the service or enforcing specific access policies. Despite the server on the onion service's side being active, the error response suggests that the server is not expecting a response or is being obstructed by a firewall. Ports that result in a "connection refused" error when attempting to establish a connection to an onion service can be referred to as "Filtered Ports."

"An onion service is considered online when a client requests a connection with the service, and it is reachable and responsive within the Tor network." In other words, an onion service is said to be online when the following sequence of steps is successful:

- Descriptor Availability: A fundamental criterion for an onion service to be online involves successful retrieval of descriptors by the Tor client, encompassing essential information like introduction points and public keys.
- Introduction Phase: The Tor client establishes a connection to at least one introduction point specified in the descriptors.
- Circuit Establishment: The onion service connects to the rendezvous point by verifying the one-time secret string received during the introduction phase.

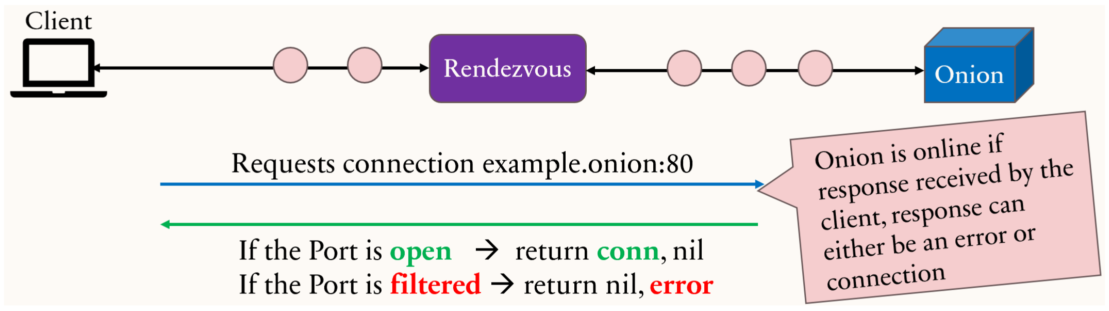

Client
Rendezvous
Onion
Requests connection example.onion:80
If the Port is open → return conn, nil
If the Port is filtered → return nil, error
Onion is online if response received by the client, response can either be an error or connection

Figure 5.4: Online Scenario

- Responsive Functionality: The service effectively handles and responds to client requests, demonstrating its ability to fulfil defined functionalities.

5.4.2. Status Check of Onions

In the context of our research methodology, the initial verification of onion service status is the foundational step preceding any port scanning activities. This procedural choice is driven by several key justifications deeply rooted in the optimization of our approach. Firstly, confirming the operational status of onion services prior to scans serves as a pragmatic resource management strategy. By discerning the availability of services upfront, we strategically allocate computational resources only to active and accessible targets, thereby maximizing the efficiency of our scanning endeavours.

This approach is particularly pertinent in large-scale scanning operations, where judicious resource utilization is imperative for feasibility and scalability. Moreover, beyond resource optimization, our commitment to ethical research practices underscores the importance of verifying service availability. By abstaining from scanning inactive or inaccessible services, we uphold principles of network integrity and responsible engagement within the Tor ecosystem. Algorithm 1 represents the pseudo-code on how onions are classified into different statuses based on the response received. It commences by initializing the status variable as an empty string and subsequently endeavours to establish a connection using the GetNetworkConnection function, which is represented by Algorithm 2. Upon encountering an error during the connection attempt, the algorithm examines the error message to determine the cause. If the error message indicates “host unreachable”, it categorizes the onion service as “offline”. Alternatively, if the error message contains terms like “TTL” or “timed”, it labels the service as “temporarily_unavailable”. It dissects the error message for other unforeseen errors, extracting the last two words and concatenating them with an underscore to formulate a distinct status. In the absence of any errors, it concludes that the onion service is “online”. Finally, the algorithm returns the determined status to indicate the service’s operational state clearly.

Now, we would like to know the distribution of response time of different statuses. It helps in evaluating the performance of onion services. Understanding the response times under different status codes makes it possible to identify areas where optimization may be needed. For example, slow response times for certain status codes could indicate bottlenecks or inefficiencies that must be addressed.

A systematic approach was adopted to mitigate the bias introduced by fluctuating network conditions. A random sample comprising 5 sets, each containing 20,600 onions, was scanned at five distinct time points. This methodology, totalling 103,000 checked onions, ensures a comprehensive and unbiased assessment of response times across various network conditions. The response time distributions from all five experiments were aggregated to analyse the collected data effectively. Visualization techniques, including box plots and kernel density estimate plots, were employed to clearly and concisely represent the response time distribution. These visualizations offer valuable insights into the variability, central tendency, and outliers within the response time data, facilitating informed decision-making and performance optimization efforts for onion services within the Tor ecosystem.

5.4.3. T.U Onions

In our exploration of onion service statuses, the behaviour of ’T.U’ onions emerges as particularly intriguing due to its inherent uncertainty. Such onions may exhibit this status for various reasons, including

Algorithm 1: Get Status of Onion Services

```
Input:
onionService: string (the onion service to connect to)
port: int (the port number)
proxyAddress: string (the address of the Tor SOCKS proxy)
timeout: time.Duration (timeout duration for the connection)
Output: status
begin
    status ← “”;
    connection, err ← GetNetworkConnection(onion, port, proxyAddress, timeout);
    if err $\neq$ nil then
        if err contains “host unreachable” then
            status ← “offline”;
        else if err contains “TTL” or err contains “timed” then
            status ← “temporarily_unavailable”;
        else if err contains “connection refused” then
            status ← “online_but_filtered_port”;
        else
            s ← Split err by spaces;
            status ← Join last two words of s with “_”;
    else
        status ← “online”;
    return status;
```

Tor network issues, rendezvous unavailability, or server-side congestion. To gain deeper insights into the behaviour of these onions and discern the conditions under which they transition from this state, we employed a systematic monitoring approach.

The experiment was conducted as follows: We initiated the process by randomly selecting a dataset of 10,000 onions, which were then processed using the OnionScan tool precisely at 10:00:00 hours. The tool systematically checked the status of each onion, flagging those identified as ’T.U’ and placing them in a dedicated queue for further monitoring. This queue system was designed with meticulous attention to detail, monitoring the status of the flagged onions at regular intervals of 5 minutes over a duration of two days. We repeated this experiment five times, with each iteration commencing at 10:00:00 and spanning over two weeks. For each repetition, a new batch of 10,000 onions was randomly selected.

The rationale behind conducting multiple iterations stems from the inherent unpredictability of the Tor network. Factors such as downtime in Tor relays or increased server loads on the OnionScan tool due to concurrent usage by multiple users can influence the results. By conducting multiple experiments, we aimed to account for these potential disruptions and ensure the robustness and reliability of our findings.

5.5. Validation

In the validation section, we comprehensively assess the port scanning tool to ensure its accuracy and reliability in detecting the status and open ports of onion services within the Tor network. This validation process encompasses evaluating the tool’s port scanning capabilities and the initial step of checking the status of onions before subjecting them to the port scanning module. This integrated approach allows us to validate the tool’s ability to effectively handle various states of onion services: online, offline, and T.U. We try to emulate different scenarios of the availability of onions.

5.5.1. Onion Services Setup

The process of setting up an onion service involves several steps. Firstly, the Tor package is downloaded from the official website or using package managers available for the operating system. Tor is configured to function as a hidden service upon installation by editing the torrc configuration file. Within this file, specific ports are designated for the service to listen on and any access controls desired. For instance:

```
HiddenServiceDir /var/lib/tor/BitcoinDoubler/
HiddenServicePort 80 127.0.0.1:80
```

Subsequently, the Tor service is restarted using the command "`sudo systemctl restart tor`", initiating the generation of a public/private key pair for the onion service. This key pair forms the basis of the cryptographic link between the server and Tor clients. The resulting onion address is stored at "`/var/lib/tor/BitcoinDoubler/hostname`", serving as the unique identifier for the onion service.

After generating the keys, the web server is configured to serve content over the Tor network. This involves specifying the onion hostname and port within the server configuration, ensuring accessibility via Tor. Additionally, it may be prudent to implement SSL/TLS encryption to safeguard communications between clients and the server. By meticulously following these steps, the onion service is effectively set up, providing a secure and anonymous platform for content delivery within the Tor network. The availability of the onion can be verified by accessing it through the Tor Browser. This procedure was repeated 20 times to set up a total of 20 onions.

5.5.2. Status Check of Onions

An interesting pattern was discovered while checking the status of onions: onions often shift to a T.U state in two scenarios. The first scenario may arise when the onion's server experiences a temporary outage or connectivity issues within the Tor circuit. Alternatively, the second scenario occurs when the onion service descriptors persist in the distributed hash table even after the onion service has been offline. This can happen because it was not properly deregistered from the Tor network. In such instances, initial steps to access the onion service may succeed; however, the final step of establishing a 6 hop-circuit fails because the onion service is offline and cannot connect to the rendezvous point, giving a connection timeout error. To verify this, two experiments were conducted to check the status of the onion services: proper deregistration by notifying the Tor network and deregistration without notifying..

5.5.2.1. Onion Service taken offline by notifying the Tor network

Deregistering an onion service from the Tor network involves a systematic process. First, administrators must access the torrc configuration file in the /etc/tor/ directory. Within this file, the configuration specific to the onion service must be identified and removed, including its hostname and port bindings. Additionally, any access controls or authentication mechanisms associated with the onion service should be disabled or revoked. Once these configurations are modified, the Tor service must be restarted for the changes to take effect, usually accomplished through the sudo systemctl restart tor command on Linux systems. Following the restart, the onion service is effectively deregistered from the Tor network, ensuring that it no longer participates in the network or provides access to its services.

The above procedure is followed for all twenty onions, and when the tool checked the status, it was shown offline. The reason is that, when the onion was deregistered following the above procedure, the onion service descriptors will be removed from the Distributed Hash Table, resulting in a scenario shown in Figure 5.2a.

5.5.2.2. Onion Service taken offline without informing tor network

Taking an onion service offline without properly deregistering it from the Tor network can lead to connection timeout errors for users attempting to access the service. To investigate this impact, an experiment was designed to simulate scenarios where onion services are taken offline without notifying the Tor network. The experiment aimed to elucidate the Tor network's and client connections' behaviour when attempting to access offline onion services under such circumstances. The experimental setup involved creating a controlled environment where a set of 20 onion services were intentionally taken offline by shutting down the servers hosting them without proper deregistration from the Tor network. Subsequently, when attempts were made by the tool to check the status of these onion services, it was shown as T.U.

This action effectively removes the service from availability, rendering it inaccessible to users. However, clients trying to establish connections face timeouts as the onion service descriptors persist in the distributed hash table. The Tor network persists in routing incoming requests to the now-offline onion service, leading to failed connection attempts and eventual timeout errors for users. Consequently, despite the service being offline, it remains referenced within the Tor network, causing connection attempts to result in timeouts instead of immediate rejection.

5.5.3. Port Scanning

An experimental setup was designed utilizing a dataset of custom onion services configured to have varying port configurations. The setup involved opening different ports on some onion services while closing different ports on others. Specifically, 20 custom onion services were created, each with a unique combination of open and closed ports. The ports ranged from commonly used ports such as HTTP (80), HTTPS (443), SSH (22) and FTP (21) to less common ports IRC (6667) to ensure comprehensive testing of the port scanning tool's capabilities.

The experimental procedure began by running the port scanning tool on the dataset of custom onion services. The tool was configured to scan for open ports within the specified range and record the results for each onion service. Subsequently, the obtained results were compared against the known port configurations of the custom onion services to evaluate the accuracy of the port scanning tool.

Upon completion of the experimental procedure, it was observed that the port scanning tool achieved a remarkable accuracy rate of 100%. The tool successfully identified and reported the status of all open ports within the custom onion services, accurately distinguishing between open and closed ports. Additionally, no false positives or false negatives were encountered during the scanning process.

The experimental results validate the port scanning tool's capabilities and effectiveness in accurately detecting open ports within onion services. The achieved accuracy rate of 100% underscores the tool's reliability and robustness, indicating its potential utility for cybersecurity professionals, researchers, and other stakeholders operating within the Tor network. Furthermore, the experiment highlights the importance of thorough testing and validation to ensure the efficacy and reliability of cybersecurity tools in real-world scenarios.

6 Implementation

This chapter delves into the technical details of implementing our port scanning tool designed for the dark web, developed based on the robust design principles outlined previously. Building on the methodology, we explore the detailed processes and algorithms that enable our tool to interact effectively with the Tor network. This includes an in-depth examination of how requests are sent through Tor's SOCKS protocol, the specific algorithm for identifying open ports, and the experiment setup to assess scalability and optimization. Additionally, we highlight the new features introduced in the extended version of the Onionscan tool, which enhances its functionality and performance in dark web investigations. We ensure our tool is efficient, accurate, and extensible by grounding our implementation in the established design principles. Through this comprehensive implementation discussion, we aim to clearly understand the technical foundations underpinning our tool's capabilities and reliability.

6.1. Port Scanning

Upon confirming the status of the onion services, we proceed to the port scanning phase, aiming to identify the open ports of the online onion services. This subsection elucidates the methodology for establishing a connection and communication with the Tor network using the SOCKS Protocol. Additionally, we detail the algorithmic approach utilized for detecting open ports within the Dark Web environment. To further assess the feasibility and efficacy of our port scanning methodology, a series of targeted experiments were meticulously designed and executed, providing valuable insights into the practical application and performance of conducting a comprehensive port scan.

6.2. Tor's SOCKS Protocol Integration

Communication with the Tor network is essential to facilitate user interaction with Onion services. Users typically employ a local software component known as an Onion Proxy (OP) to manage tasks such as fetching directories, establishing circuits across the Tor network, and managing connections from user applications. These onion proxies are designed to accept TCP streams and efficiently multiplex them across established circuits. On the opposite end of these circuits, onion routers connect to the requested destinations and relay data accordingly. Users utilise the Tor onion proxy to initiate connections with the Tor network. However, direct communication between client applications and the Tor proxy isn't supported. The Tor network supports the SOCKS protocol to bridge this gap, enabling seamless communication with the Tor proxy. The SOCKS proxy serves as a mediator in this integration, facilitating communication between client applications and the Tor network. The SOCKS protocol presents a standardized interface for TCP proxies. Client software initiates a TCP connection to a SOCKS server and requests a TCP connection to a specified address and port. The SOCKS server then establishes the connection and communicates the outcome—success or failure—back to the client. Once the connection is established, the client application customarily utilizes the TCP stream. Subsequently, the Tor Onion proxy forwards the client's request to the Tor network, enabling secure and anonymous communication with Onion services.

A detailed explanation of each step involved in this integration: The client application initiates a connection to the local SOCKS proxy server, specifying the destination address and the desired protocol (TCP). The SOCKS proxy listens for incoming requests and is an intermediary between the client and the Tor onion proxy. Upon receiving the client's request, the SOCKS proxy encapsulates the Tor onion proxy dialer functionality. This encapsulation allows the SOCKS proxy to route the client's traffic through the Tor network, leveraging the anonymity and security features provided by Tor. The encapsulated Tor onion proxy dialer within the SOCKS proxy forwards the client's request to the Tor network. This step involves establishing a connection to a Tor node and routing the client's traffic through the Tor network's decentralized infrastructure. The Tor network receives the forwarded request from the SOCKS proxy and routes the client's traffic through multiple Tor nodes. Each node in the Tor network decrypts and re-encrypts the data, obscuring its origin and ensuring secure transmission through a randomized pathway. After processing the client's request, the Tor network fetches the desired data from the destination server. The response is then transmitted back through the secure and anonymous Tor network route, passing through the same set of Tor nodes. The Tor onion proxy dialer within the SOCKS proxy receives the response from the Tor network. The SOCKS proxy then relays this response to the client application, completing the communication loop.

Thus, integrating the SOCKS protocol within the Tor ecosystem is vital due to its widespread adoption among applications [29]. Unlike the Tor protocol, which is specific to the Tor network, applications are more familiar with communicating via SOCKS. Leveraging the SOCKS interface simplifies the integration process for applications, as they can utilize the existing proxy interface without needing custom integration. Additionally, SOCKS is preferred due to its simplicity and versatility. Protocols like SOCKS4a and SOCKS5 further enhance compatibility by supporting direct connections to hostnames, mitigating the risk of DNS leaks.

Algorithm 2: Network Connection using SOCKS Protocol

```
Input:
onionService: string (the onion service to connect to)
port: int (the port number)
proxyAddress: string (the address of the SOCKS proxy)
timeout: time.Duration (timeout duration for the connection)
Output:
net.Conn (network connection)
error (if any error occurs during connection)
Step 1: Convert the port number to a string representation.;
Step 2: Create a SOCKS5 dialer using the provided proxy address.;
Step 3: Check for any errors during the creation of the SOCKS5 dialer.;
  if an error occurs then
      Return nil for the connection and the error;
  else
      Step 4: Dial a TCP connection to the specified onion service and port using the
        SOCKS5 dialer.;
      Step 5: Check for any errors during the connection establishment.;
        if an error occurs then
            Return nil for the connection and the error;
        else
            Step 6: Set the deadline for the connection using the provided timeout
              duration.;
            Step 7: Return the established connection and nil for the error.;
        end
  end
```

Now, let's delve into the methodology used to communicate with the Tor network via the SOCKS protocol as shown in Algorithm 2:

- Initialization: The process begins by initializing a SOCKS proxy client within the client application. This involves creating a SOCKS proxy client object or utilizing a library that supports SOCKS proxy.
- Configuration: The SOCKS proxy client is configured with the necessary parameters, including the address and port of the SOCKS proxy provided by the Tor client.

- Connection Establishment: With the SOCKS proxy client configured, connections to the Tor network can be established. The client specifies the address and port of an onion service when initiating the connection.
- Communication: Once the connection is established, the client application can communicate with the desired destination within the Tor network. The SOCKS proxy client encapsulates and forwards the application's network traffic through the Tor network, ensuring anonymity and privacy.
- Data Transfer: Data is transferred between the client application and the destination within the Tor network over the established connection. The SOCKS proxy client transparently handles the data transfer process, encrypting the data multiple times (onion routing) as it traverses the Tor network.

6.3. Scanning Onion Services for Open Ports

The OnionScan tool is designed with a primary focus on detecting open ports of onion services rather than determining the status of closed ports, as detailed in Section 5.4.1.3. All active onion services undergo the port scanning procedure outlined in Algorithm 3. The port scanning function takes the following input parameters: Text file with the list of onions to be scanned, Tor proxy address, Port range to be scanned (optional parameter: by default, it scans first 1000 ports). The variables in the algorithm are:

- **openPorts:** A string to store the list of detected open ports.
- **status:** A string to store the status of the Onion service.
- **checkPort:** The port on which the status needs to be checked initially before feeding the onion to the port scanning loop is represented as X1 in the algorithm.
- **maxConcurrent:** The maximum number of concurrent port scans, defined by osc.PortConcurrency. (In the thesis we use X2=200 as the limit considering the maximum load the server running the tool can handle)
- **semaphore:** created to limit the number of concurrent goroutines to maxConcurrent.
- **wg (sync.WaitGroup):** Initialized to keep track of all goroutines.

In the ScanProtocol function, semaphore, mutex and waitgroup are used to efficiently and safely scan a range of ports on onion services. Semaphores are used to limit the number of concurrent port scans. By setting maxConcurrent to 200, the function ensures that no more than 200 port scans are performed simultaneously. This prevents system overload and optimizes resource usage, allowing the scanning process to run smoothly without overwhelming the system. The sync.WaitGroup is crucial for synchronizing the completion of all goroutines. By adding a counter before spawning each goroutine and decrementing it upon completion, the main function can wait for all scans to finish. This guarantees that all ports are scanned before proceeding to the next steps, ensuring the accuracy and completeness of the scan results. The sync.Mutex ensures thread-safe operations when updating shared variables like openPorts. Concurrent writes to openPorts without synchronization could lead to race conditions and inconsistent data. The mutex locks access to openPorts during updates, maintaining data integrity and preventing conflicts between goroutines. These concurrency primitives together provide a robust and efficient framework for parallel port scanning, ensuring data integrity and system stability while maximizing the efficiency of the scan process.

The algorithm employs a for loop to iterate over the specified port range, utilizing Go routines to execute port scans concurrently, thereby enhancing the efficiency of the scan process. Currently, 200 ports run concurrently at a time. Within each iteration of the loop, a TCP connection request is dispatched to the onion address and the corresponding port number, as demonstrated in Algorithm 2. If the connection request results in no errors and the connection variable isn't nil, it indicates that the port is open and can accept incoming connections. Finally, the identified open ports are recorded in the database for subsequent analysis and evaluation of the scan results.

Having established the methodology for port detection, our subsequent objective shifts towards identifying the most frequently used protocols. To achieve this, we conducted a series of scans across various port ranges, including 1-1000, 1-10000, and a comprehensive scan of up to 65536 ports.

Algorithm 3: Port Scanning Onion Services

```
Input: onion, onionId, osc
Output: None
begin
    openPorts ← “”;
    status ← “”;
    checkPort ← X1;
    maxConcurrent ← X2;
    Initialize sync.WaitGroup $wg$;
    Create semaphore with maxConcurrent;
    status m,sd/k’leijo32i90← getStatus(hiddenService, checkPort, osc.TorProxyAddress,
    osc.Timeout);
    if status == “online” or status == “online_but_filtered_port” then
        foreach port in defaultPorts do
            wg.Add(1);
            semaphore ← acquire lock;
            Spawn thread to check port status;
            if connection is successful then
                Add port to openPorts;
        wg.Wait();
        close(semaphore);
    LogInfo(onion, openPorts);
    db ← InitDB();
    if Error initializing database then
        Print error message;
        return;
    if Error inserting/updating to database then
        Print error message;
    Close db;
```

6.4. Scalability and Optimization Assessment of Port Scanning

Understanding the time required for a full scan of an onion service is crucial, as a complete scan for every service isn’t always feasible due to time constraints and computational resources. Thus, optimizing the port scanning methodology becomes essential. To refine our approach, we conducted scans on a randomized sample of 100 onion services, assessing the time taken for port scanning across different port ranges, including 100, 1000, 2000, 5000, 10000, 15000, 20000, 30000, 40000, 50000, and 65536 ports. Subsequently, the time distributions for these scans were visualized to analyze the practicality and efficiency of conducting port scans across various port ranges. This analysis aids in determining the optimal port range for efficient scanning while balancing thoroughness and resource utilization.

6.5. Extended Onionscan for Port Scanning

In implementing the port-scanning methodology, a tailored selection of tools and technologies has been orchestrated to ensure efficiency, extensibility, and depth of analysis. The initial step involves leveraging a Go script, specifically designed to interact with the Dark Web Monitor API, facilitating the retrieval of the onion services dataset. The port-scanning module is constructed upon the foundational framework of Onionscan [30]. The tool is extended to address specific limitations inherent in the standard Onionscan tool, enhancing its functionality to suit the research requirements. Notably, the extension enables scanning a broader range of ports, either default or specified range, surpassing the default capability limited to the top 10 common ports. This enhancement offers a more comprehensive exploration of the diverse services within the Dark Web environment, facilitating a deeper understanding of its inner workings.

In addition to port scanning, the modified tool incorporates functionalities for monitoring the status of onions, categorizing them as online, offline, or T.U. A queue system enables the continuous monitoring of onions in the latter two categories, ensuring timely detection of changes in their status. Furthermore, the tool allows the input of a list of onions to be scanned, offering flexibility in targeting specific onion services for analysis.

To optimize performance and efficiency, the tool supports the utilization of goroutines. Goroutines in Go (Golang) provide a lightweight way to achieve concurrency. Each goroutine is a function that runs concurrently with other goroutines in the same address space. Launching a separate goroutine for each port or onion service being scanned allows the tool to perform multiple scans in parallel, significantly reducing the total scan time. This concurrent execution capability enhances the scalability and effectiveness of the scanning process, allowing for swift and thorough exploration of the Dark Web landscape.

Moreover, integrating MySQL database functionality seamlessly complements the scanning tool, facilitating the systematic storage of scan results. This refinement contributes to a well-organized and structured database, laying the groundwork for subsequent analyses and providing a comprehensive repository of Dark Web intelligence. For data analysis and visualization purposes, a Python notebook is chosen.

7
Results

In this chapter, we present the results of our investigation to provide valuable insights into the characteristics and behaviours of onion services on the dark web, utilizing the specialized port scanning tool developed in this research. We begin by analyzing the average response times of onion services with different status codes. Of these onions, we are particularly interested about T.U onion services. So, these onion services are monitored further to find out the dynamics of such onion services. Following this, we conduct a thorough post-data analysis, examining the distribution of open ports, the prevalence of onions with and without port 80 open, and grouping onions based on the number of detected open ports. We also explore the time distribution of port scanning and perform a detailed analysis of the services identified, including a temporal analysis to observe service trends over time. Additionally, we conduct a correlation analysis to understand the relationships between individual ports and service-based groupings. Finally, we discuss the optimized port scanning methodology, highlighting when to revisit the onion services and the implementation of continuous integration to enhance the robustness and efficiency of our approach. This comprehensive results section aims to provide valuable insights into the characteristics and behaviours of onion services on the dark web.

7.1. Dynamics of Onion Services

This section encompasses a diverse range of analyses, each contributing unique insights into the operational dynamics and performance metrics of Onion Services.

7.1.1. Average Response Times of Different Status Codes

Status Code	count	mean	std	min	25%	50%	75%	max
online	92,076	6.03	5.71	0.14	1.31	4.40	8.70	28.00
offline	10,000	16.24	8.20	2.73	12.70	16.54	21.19	63.97
T.U	924	120.02	0.48	119.00	120.00	120.00	120.11	121.00

Table 7.1: Response Time Statistics of Onions in Different Status Categories

The time taken to get a response when a connection request is made to an onion service was measured for a random sample of 103,000 onions in total. Figure 7.1 shows a boxplot of these response times. Further, 7.1 provides comprehensive statistics of the response times observed across different categories of onion statuses.

For onions categorized as “offline,” the data reveals that 10,000 were identified in this state. The average response time for these onions is approximately 16.24 seconds, with a standard deviation of 8.20 seconds, indicating some variability in the response times. The minimum response time recorded is 2.73 seconds, while the maximum is 54.70 seconds. Notably, the median response time, representing the data’s midpoint, stands at 15 seconds, with 25% of onions responding in less than 10.29 seconds and 75% responding within 20.39 seconds.

On the other hand, onions categorized as “online” constitute a larger portion, totalling 92,076 onions.

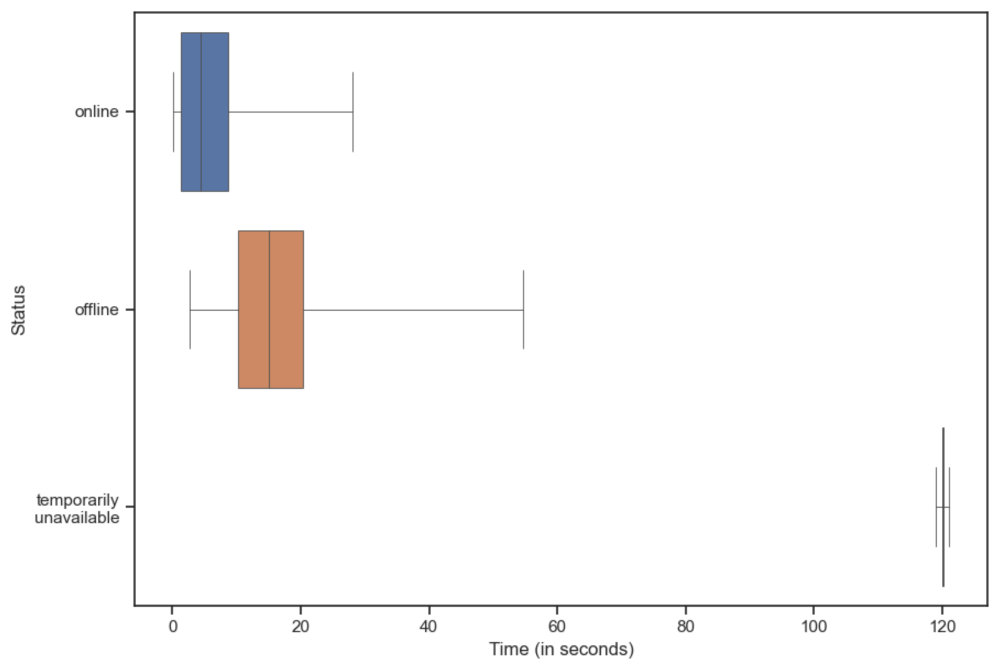


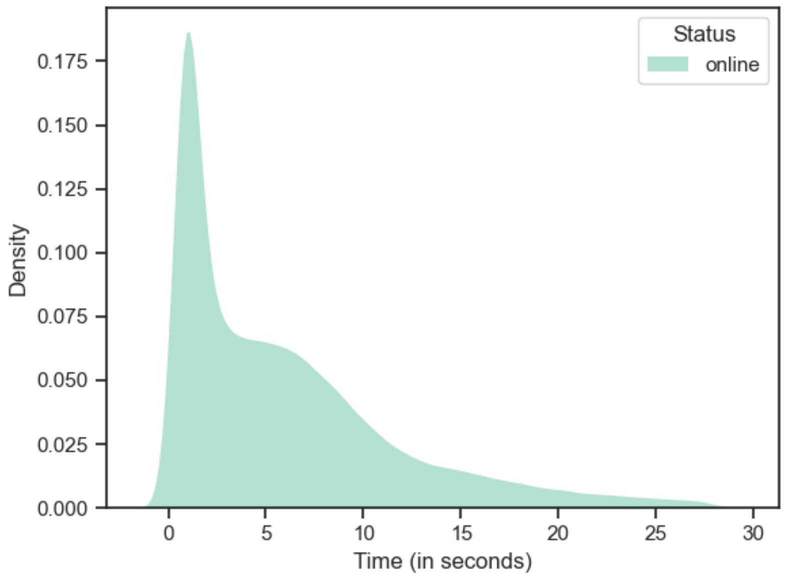
Figure 7.1: Boxplot representing response times of onions under different status categories

These onions exhibit comparatively shorter response times, with an average of approximately 6.03 seconds and a standard deviation of 5.71 seconds. The distribution of response times for scanned onions shows a similar pattern, with 25%, 50%, and 75% of onions responding within 1.31, 4.40, and 8.70 seconds, respectively. The response times range from 0.14 to 28.00 seconds, capturing the variability in accessibility among these onions.

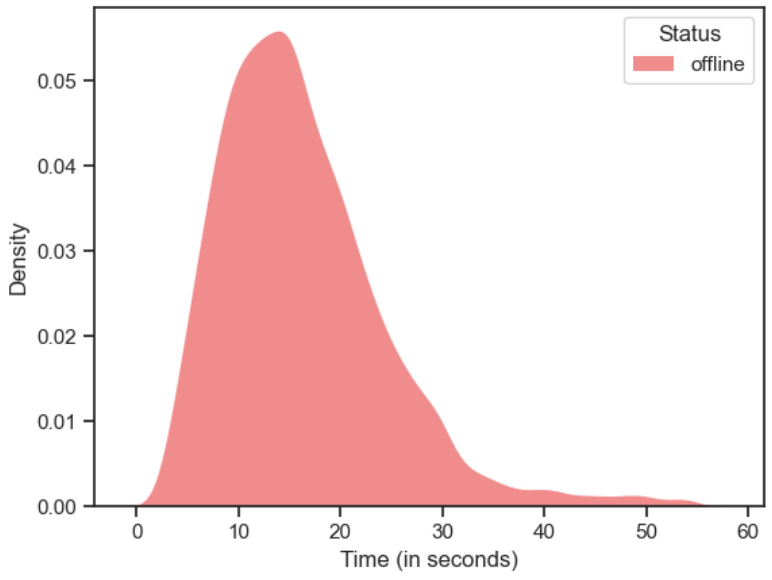
In contrast, the "T.U" category comprises 924 onions experiencing transient unavailability. The statistics for this category reveal notably higher response times, with an average of approximately 120.02 seconds and a standard deviation of 0.48 seconds. Despite some onions showing relatively shorter response times, the median and the maximum response time remain consistent at 120 seconds. A timeout of 180 seconds was set while making an HTTP connection, but here, we observe that the distribution lies almost exactly at 120 seconds.

The timeout observed in the Tor network, typically around 120 seconds, results from a comprehensive process orchestrated by the Tor client to establish optimal circuit build timeouts for connecting with onion services. The process involves recording and analyzing circuit build times, estimating distribution parameters, and calculating timeouts based on the Pareto distribution fitting of the data. Firstly, the Tor client collects and stores circuit build times in a circular array, updating distribution parameters and recomputing the timeout after each circuit completion. The parameters for a Pareto distribution are calculated using the maximum likelihood estimator, with the distribution mode serving as the Xm parameter.

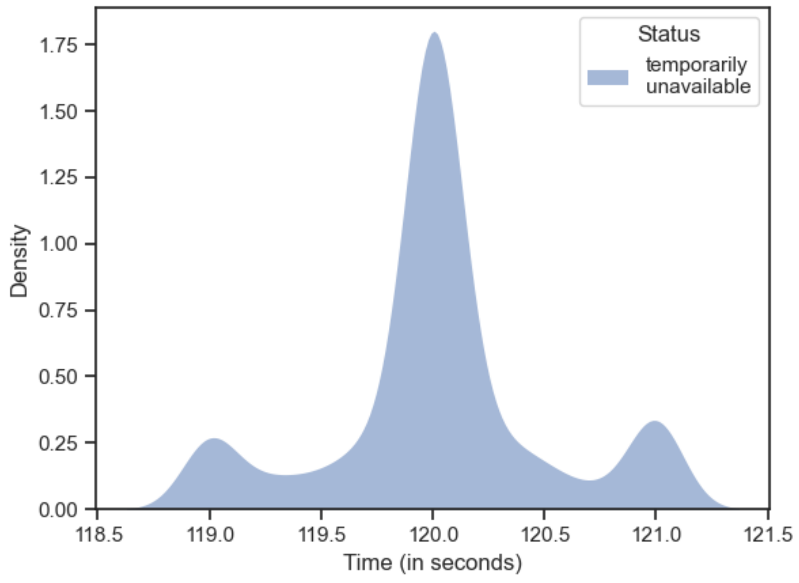
The circuit build timeout is then determined using the Pareto Quantile function, which yields the cumulative distribution function (CDF) value such that 80% of the distribution mass is below the timeout value. During the calculation of circuit build timeouts, Tor clients cap the timeout value, denoted as timeout_ms, at the maximum build time observed thus far, but at least 60 seconds. This means that if a circuit takes an unusually long to build, possibly due to relay issues, the client will not wait indefinitely but instead sets a maximum limit based on historical build times. Additionally, the close timeout, represented as close_ms, is capped at twice the maximum observed build time, ensuring that circuits are not kept open indefinitely. Since the default value of cap close_ms is twice 60 seconds, it results in a cap of 120 seconds. Therefore, the observed result of 120 seconds is consistent with the timeout mechanism described.



(a) Distribution of Online Onions



(b) Distribution of Offline Onions



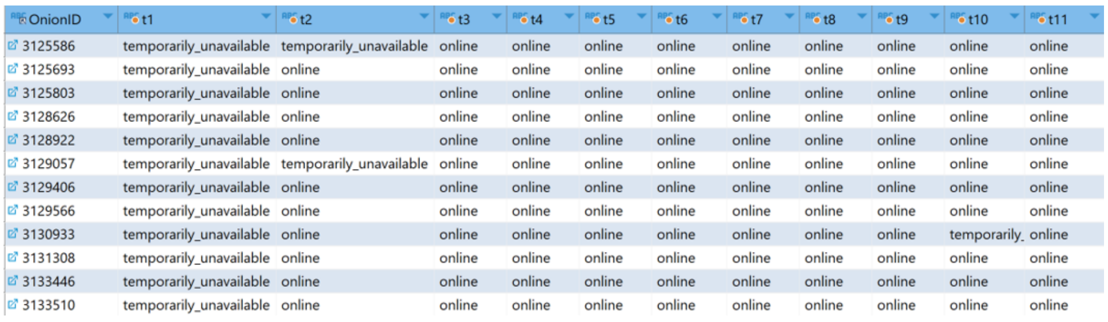
(c) Distribution of T.U Onions

Figure 7.2: Kernel Density Estimate plot of response times of different onion status categories

The kernel density estimate plots shown in Figure 7.2a, 7.2b, and 7.2c depicts the distribution of response times across different onion statuses. An interesting observation arises from the response time perspective upon analysing the plots. Onions categorized as T.U exhibit significantly longer response times than offline and online onions, aligning with our expectations. Furthermore, offline onions display slightly higher response times than online ones. Onions taking longer than 60 seconds to respond indicate potential temporary unavailability. This finding underscores the significance of response time thresholds in identifying the status of onions. As such, T.U onions present a particularly intriguing subset for further investigation.

7.1.2. Monitoring T.U onions

In this section, we outline the heuristics employed for revisiting onion services, which involves periodically checking the status of these services. Effective monitoring is crucial for maintaining an up-to-date understanding of onion service availability and behaviour. To ensure comprehensive coverage, we categorize the revisits into three main subsections: monitoring online, offline, and T.U onions. Each subsection addresses specific considerations and strategies tailored to the distinct characteristics and potential challenges associated with onions in these different states. By systematically revisiting onion services based on these heuristics, we aim to enhance the reliability and timeliness of our assessments while gaining deeper insights into onion service dynamics.



OnionID	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11
3125586	temporarily_unavailable	temporarily_unavailable	online	online	online	online	online	online	online	online	online
3125693	temporarily_unavailable	online	online	online	online	online	online	online	online	online	online
3125803	temporarily_unavailable	online	online	online	online	online	online	online	online	online	online
3128626	temporarily_unavailable	online	online	online	online	online	online	online	online	online	online
3128922	temporarily_unavailable	online	online	online	online	online	online	online	online	online	online
3129057	temporarily_unavailable	temporarily_unavailable	online	online	online	online	online	online	online	online	online
3129406	temporarily_unavailable	online	online	online	online	online	online	online	online	online	online
3129566	temporarily_unavailable	online	online	online	online	online	online	online	online	online	online
3130933	temporarily_unavailable	online	online	online	online	online	online	online	online	temporarily_	online
3131308	temporarily_unavailable	online	online	online	online	online	online	online	online	online	online
3133446	temporarily_unavailable	online	online	online	online	online	online	online	online	online	online
3133510	temporarily_unavailable	online	online	online	online	online	online	online	online	online	online

Figure 7.3: Database Records of monitored T.U onions

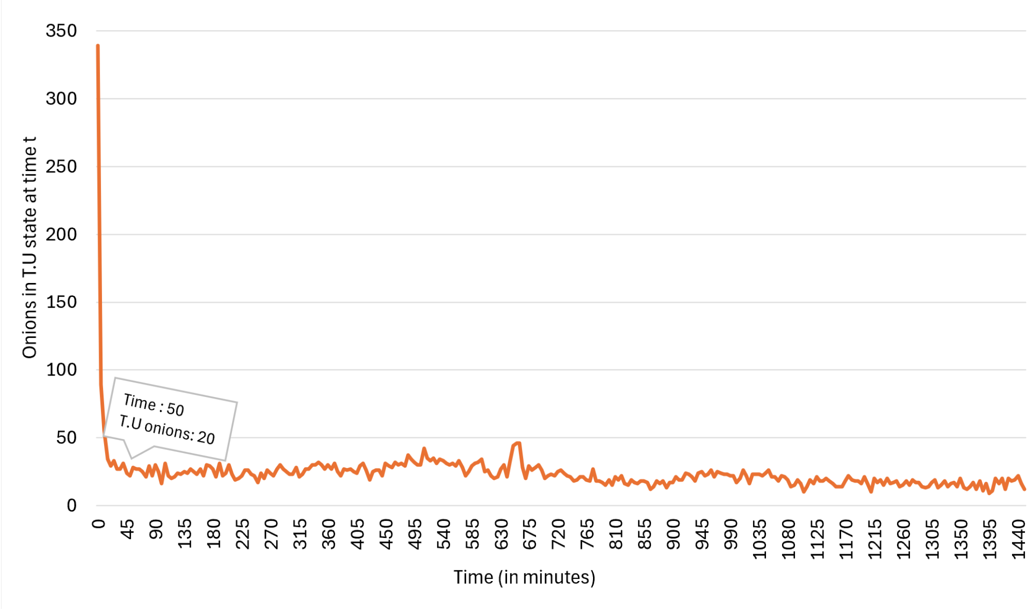
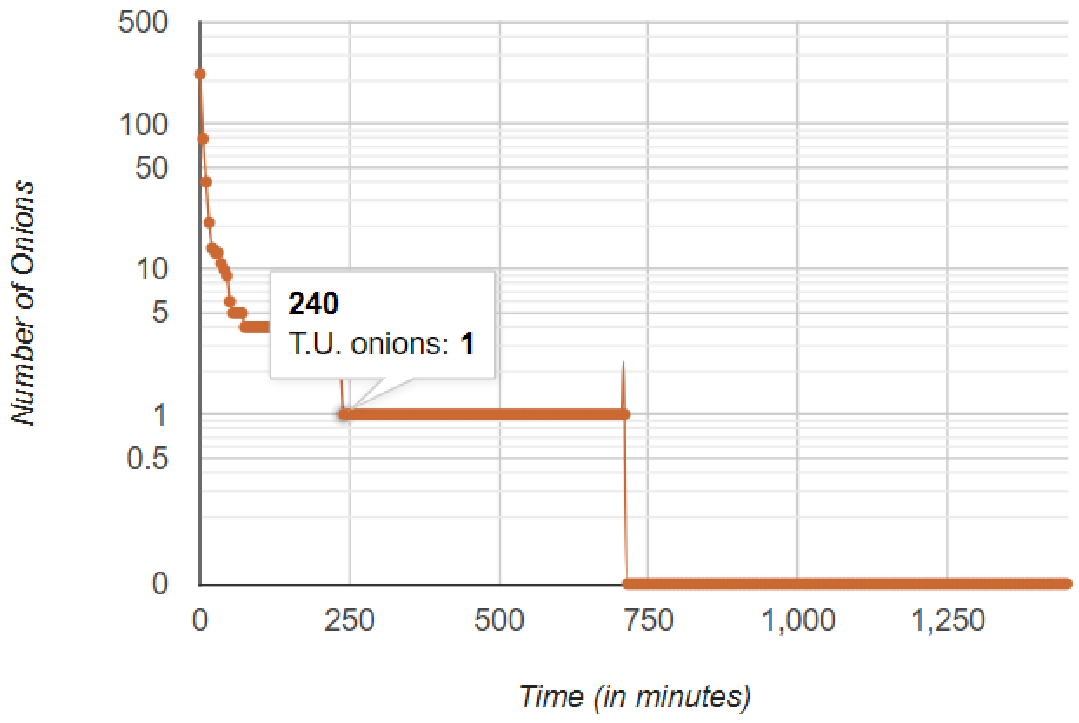
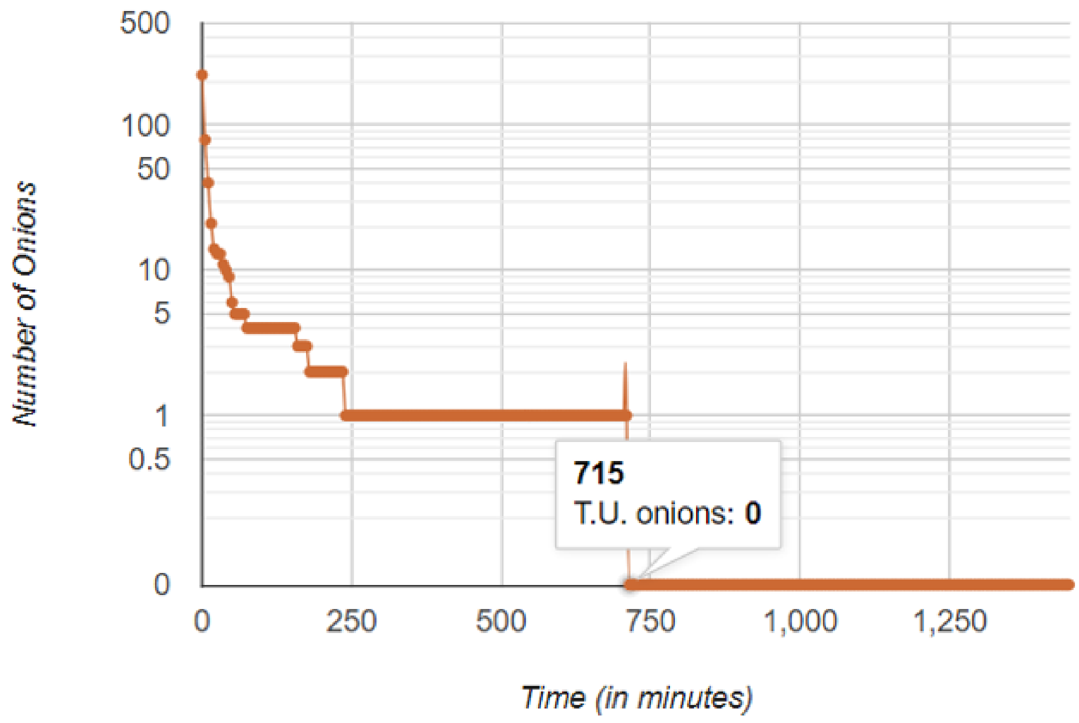


Figure 7.4: Graph representing the number of onions at T.U state at time t

Of the 50,000 onions examined, a relatively small proportion of 339 were found to be T.U and were meticulously monitored over a 2-day period at 5-minute intervals. Two distinct plots offer insights into



(a) After 4 hours



(b) After 12 hours

Figure 7.5: Graph representing how long it takes T.U onions to change their state

the statistical trends concerning the transitions of these T.U onions to online or offline states. Figure 7.3 illustrates the database records of the monitored T.U onions, spanning 290 intervals of 5-minute increments labelled as t1, t2, ..., t290. This plot meticulously tracked T.U onions that remained static in their status without transitioning to online or offline states. The primary objective was to discern the duration required for all T.U onions to undergo at least one status change.

In contrast, the second plot aimed to quantify the count of onions in various statuses at each time interval. Its purpose was to identify instances where the count of T.U onions dropped to zero, indicating a complete transition out of the T.U state.

Our analysis was further contextualized through the examination of temporal behaviours depicted in Figures 7.4, 7.5a and 7.5b. The x-axis represents discrete 5-minute time intervals in the provided visualisation, offering a comprehensive temporal representation. On the y-axis, we quantify the number of onions detected within each interval, providing a granular understanding of onion activity over time. This visualization is a dynamic tool for tracking the fluctuating presence of onions within the dataset, allowing for precise analysis of temporal trends and patterns and highlighting the timeframes within which onions transitioned statuses or remained static. Specifically, Figure 7.5a demonstrated that nearly all T.U onions experienced at least one status change within 4 hours, with only one exception. Conversely, as depicted in Figure 7.5b, all onions transitioned within 12 hours. Furthermore, Figure 7.4 showcased that after 50 minutes, only 20 T.U onions remained in the T.U state out of the initial 339, i.e., almost 95% of the onions changed their T.U state.

Considering these empirical findings, we propose heuristic guidelines for revisiting these onions. These guidelines suggest conducting follow-up checks at specific intervals, such as 50 minutes, 4 hours, and 12 hours, until each T.U onion has experienced at least one status change. Such an approach ensures a comprehensive monitoring strategy tailored to the temporal behaviours observed in the dataset.

7.2. Post Data Analysis

The dataset under analysis spans the timeframe from September 2018 to January 2024, encompassing a diverse range of onion addresses. A total of 521,565 onion addresses have been systematically scanned, as depicted in Figure 7.6. Among these, 311,613 were found to be online, while 194,306 were online, and 15,646 addresses were T.U during the scanning process.

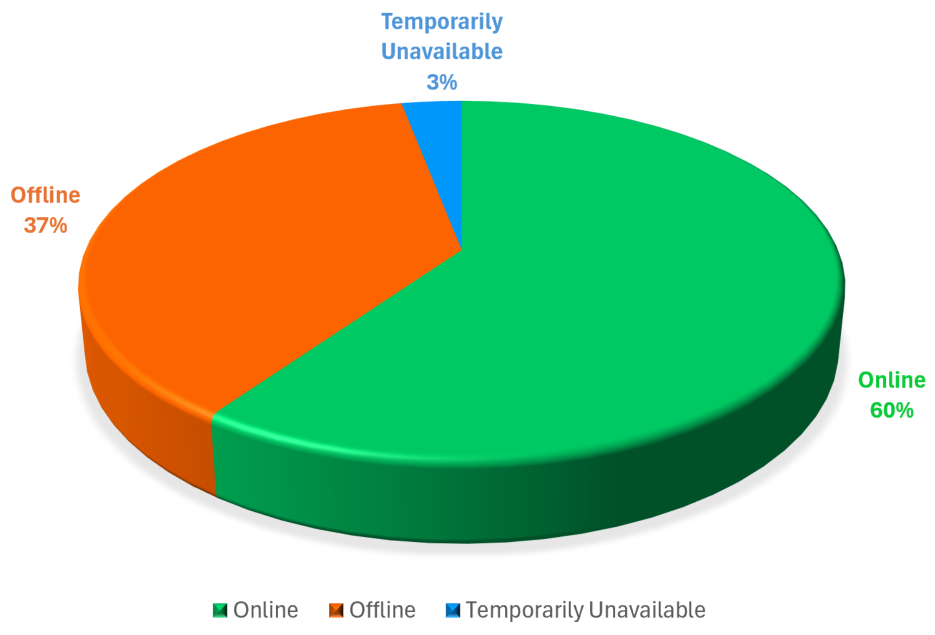


Figure 7.6: Distribution of Dataset

7.2.1. Distribution of Open Ports

Among the 311,613 online onion services, 126874 onions were scanned in the port range 1-1024, 127209 onions in the port range 1-10000, and 57530 onions scanned in the port range 1-65536.

The frequency distribution of open ports detected in these onions is depicted in Figure 7.7. The frequency distribution is represented in two plots because all the ports cannot be fit in a single graph. To refine our analysis and focus on intriguing insights, we filtered out onions with ports exclusively open on port 80, resulting in 6758 onions for further investigation. A total of 196 unique ports were identified. These unique ports are listed in Appendix A. Notably, the Lightning Network protocol (ports 9735 and 9736) exhibited the highest frequency, encompassing a mean of 2,516 onions (47%), followed by Bitcoin (ports 8333, 8332, 28333 and 28332) with 1,869 onions (29%), SSH (port 22) with 1,238 onions (21%), and HTTPS (port 443) with 887 onions (16%). Additionally, some of the noteworthy ports include Monero (ports 18081 and 18083) with 239 onions (4%), SMTP (port 25) with 87 onions (1.5%), POP3 (port 110) with 41 onions (1%), XMPP (ports 5222 and 5269) with 62 onions (0.9%) and IRC (ports 6666 and 6697) with 60 onions (0.9%).

Further investigation into the onions with open ports revealed an interesting insight. Many onions were observed without port 80 open. The following two subsections describe the two categories in detail.

7.2.2. Onions With Port 80 open

Of 298,804 online onions, 296,620 onions had only port 80 open, and 2184 onions had ports open other than port 80. To observe the most prevalent ports in this category of onions, all the onions that only had port 80 open were removed as shown in Figure 7.8. In this category, 147 unique ports were detected. The SSH service (port 22) exhibits the highest frequency with 1233 onions, followed by HTTPS (port 443) with 822 onions, alternative HTTP ports (ports 82, 84, 81) with 277 onions, SMTP (port 25) with 83 onions and POP3 (port 110) with 38 onions.

7.2.3. Onions without Port 80 open

Moreover, web crawlers predominantly target ports 80 and 443 when indexing websites, further exacerbating the invisibility of these onion services. By strategically avoiding these standard ports, these services effectively slip under the radar of conventional web crawlers, evading detection and indexing. Figure 7.9 demonstrates the common ports observed under this category. 2468 onions had 9735 ports open, a lightning protocol, and 1722 onions had 8333 ports open related to Bitcoin.

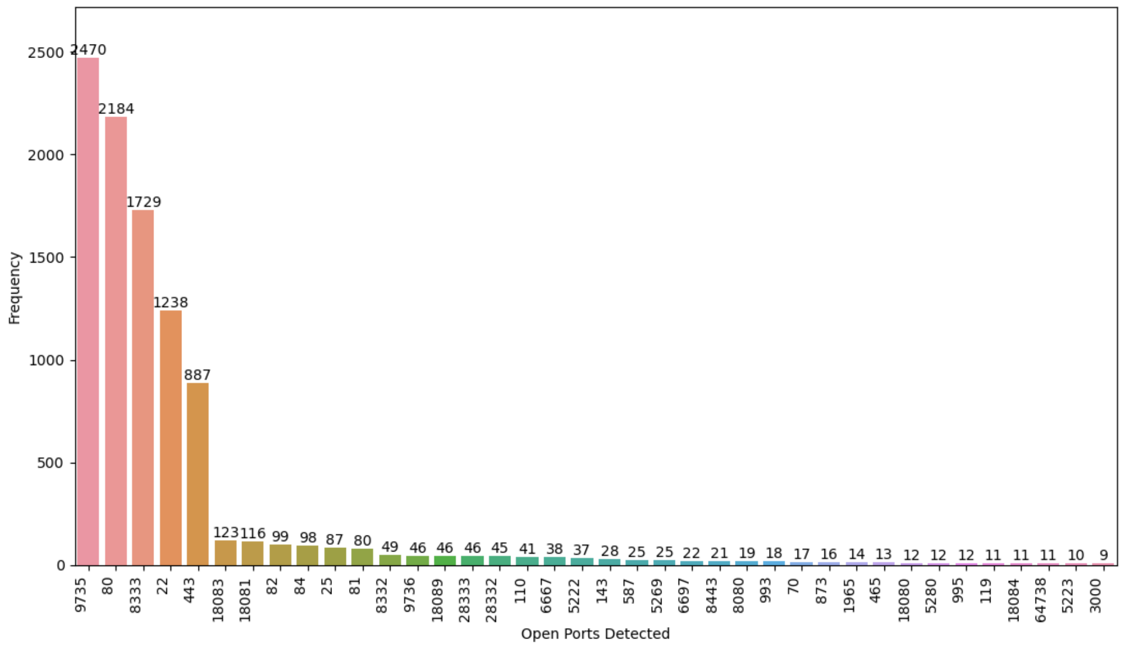
7.2.4. Grouping onions with number of open ports detected

The graph shown in Figure 7.10 illustrates the frequency of onions based on the number of open ports detected during port scanning. It reveals that most onions have a relatively low number of open ports, with 4396 onions having only 1 open port and 1757 onions having 2 open ports. The analysis revealed that the maximum number of open ports detected on an onion service was 35. Upon further investigation of onion services with 35 and 33 open ports, it was observed that they exhibited identical user interfaces when accessed through the Tor browser. The uniformity in user interface across onion services with many open ports suggests a potential pattern or standardization in the configuration or deployment of these services. It could imply that these onion services are part of a common infrastructure or network with similar functionalities or purposes. Additionally, it may indicate intentional design choices or restrictions imposed by the operators of these services to maintain consistency or enhance user experience.

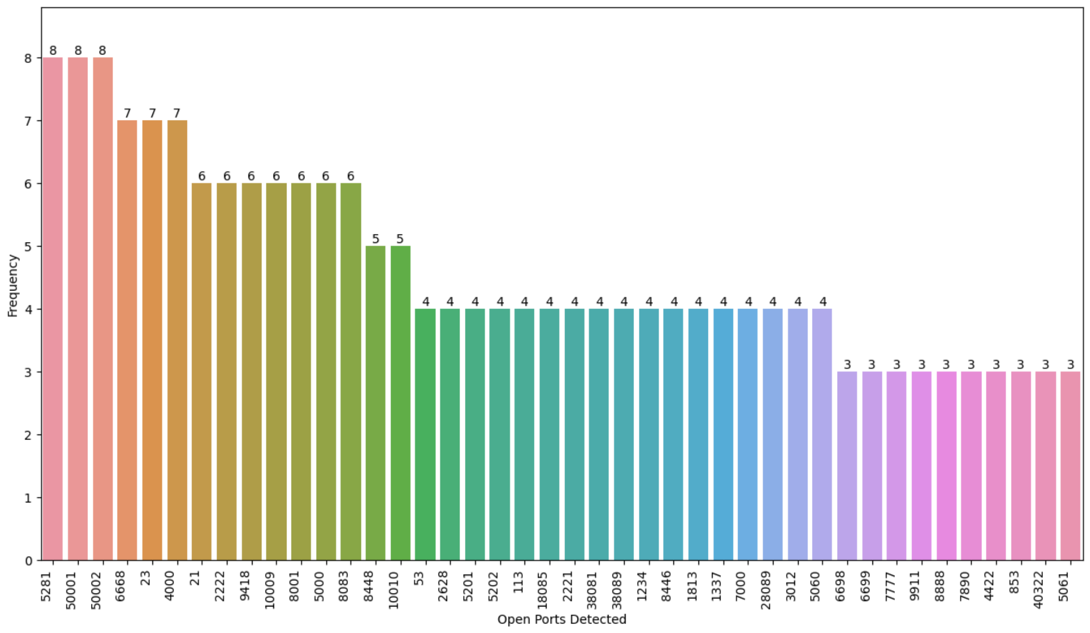
As the number of open ports increased, the frequency of onions decreased significantly, indicating that onions with more open ports were less common. The steep decline in the frequency of onions beyond a certain threshold suggests that detecting onions with many open ports is rare. This distribution provides insights into the diversity of security configurations among onions on the network, highlighting the prevalence of onions with minimal exposed services and the rarity of onions with extensive port exposures.

7.2.5. Time Distribution of Port Scanning

A boxplot illustrating the time distribution for scanning 100 onions is depicted in Figure 7.11a. On average, a full scan takes approximately 8 minutes, ranging from 6 to 20 minutes. Notably, 75% of onion services require less than 13 minutes for a complete scan. Figure 7.12 represents the distribution of time taken (in seconds) to conduct port scans across different port ranges (e.g., 1-100, 1-1000, 1-10000, etc.),

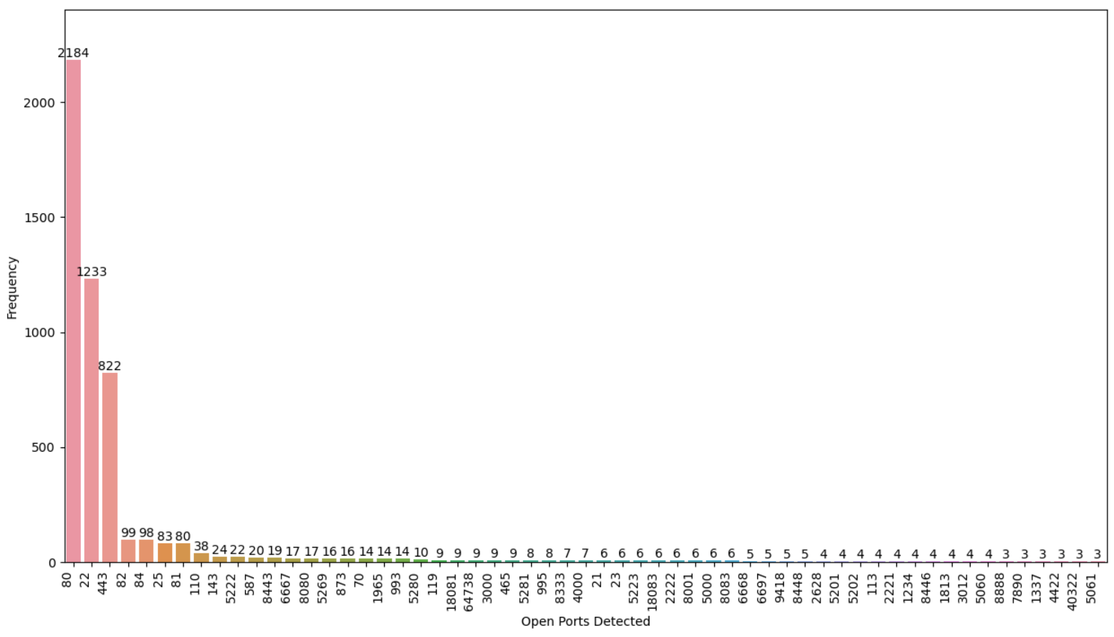


(a) Prevalent Open Ports with frequency greater than 8

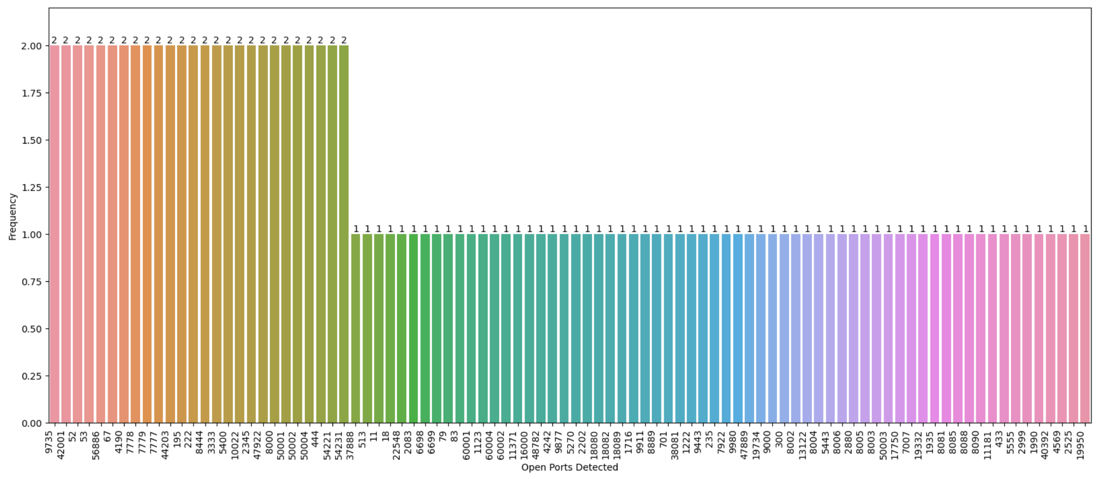


(b) Frequency of open ports with less than 9 onions

Figure 7.7: Frequency Distribution of Open Ports Detected (Excluding port 80)

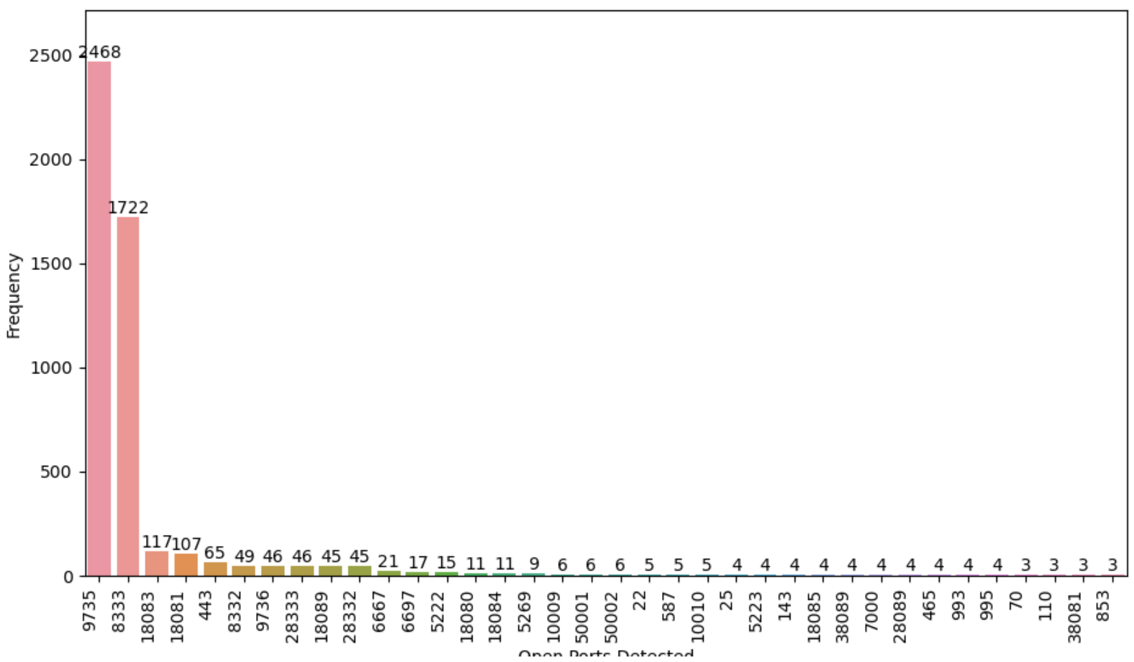


(a) Prevalent Open Ports with frequency greater than 2

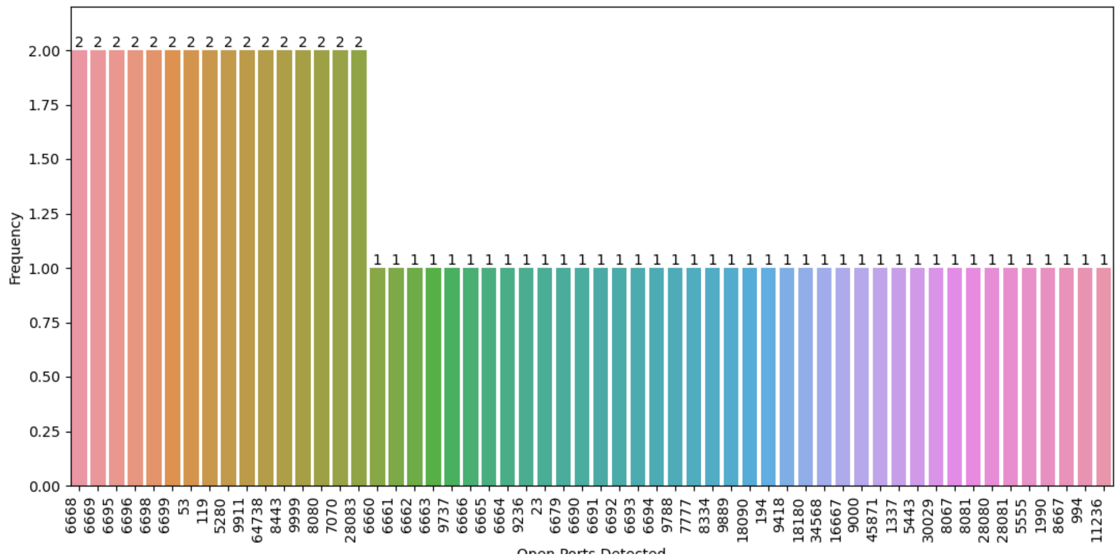


(b) Frequency of open ports with less than 3 onions

Figure 7.8: Frequency Distribution of Open Ports Detected of onions that have open ports other than 80



(a) Prevalent Open Ports with frequency greater than 2



(b) Frequency of open ports with less than 3 onions

Figure 7.9: Frequency Distribution of Open Ports Detected of onions that do not have port 80 open

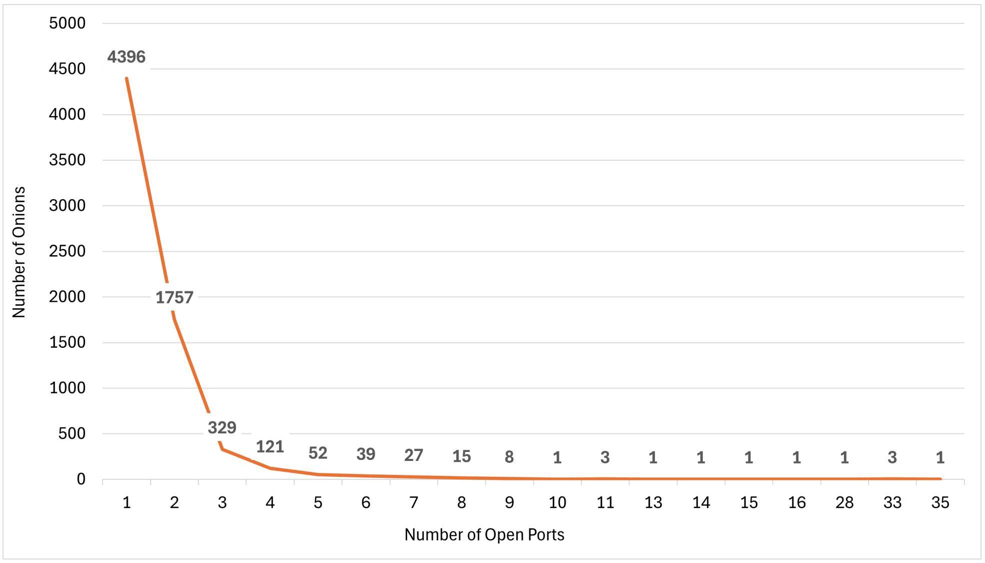


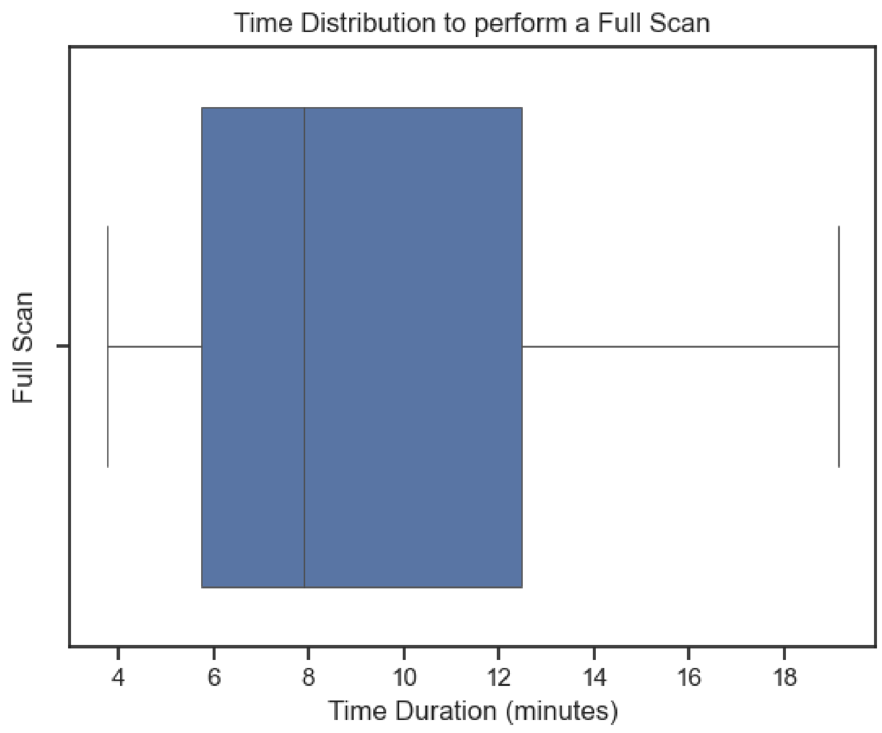
Figure 7.10: Onion Services grouped by Number of Open Ports detected

and detailed statistics are presented in the table 7.2.

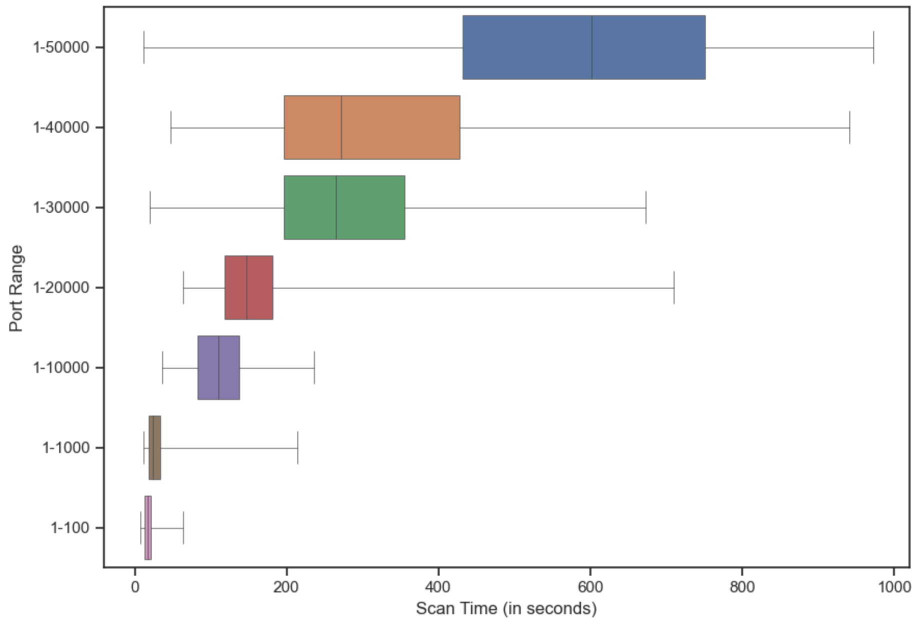
In the 1-100 port range, scans exhibit relatively swift completion times, with an average duration of 18.00 seconds and a narrow standard deviation of 8.14 seconds, indicating consistent performance across scans. The distribution of scan times within this range is relatively compact, ranging from a minimum of 7.67 seconds to a maximum of 63.97 seconds. Notably, 25% of scans were completed within 12.70 seconds or less, with a median scan time of 16.54 seconds. As the port range expands to 1-1000, scan times exhibit a moderate increase in average duration (28.28 seconds) and variability (22.05 seconds standard deviation). While minimum scan times remain relatively low at 11.32 seconds, maximum times extend to 214.15 seconds, indicative of occasional outliers. Percentile analysis reveals that 75% of scans were completed within 33.29 seconds, with a median scan time of 23.86 seconds. Expanding further to the 1-10000 and 1-20000 port ranges, scan times escalate significantly, reflecting the heightened complexity and computational demands of scanning larger ranges. In the 1-10000 range, the average scan time increases to 114.16 seconds, with a wider standard deviation of 38.11 seconds. Notably, 25% of scans were completed within 82.58 seconds, while the median scan time is 109.67 seconds. Similarly, in the 1-20000 range, the average scan time rises to 167.40 seconds, with a standard deviation of 86.11 seconds. Despite the increased variability, percentile analysis reveals that 75% of scans were completed within 181.72 seconds, indicating a consistent proportion of scans completed within a reasonable timeframe.

As the port range extends beyond 20000 to 30000 and 40000, scan times continue to escalate exponentially, underscoring the diminishing efficiency and scalability of the scanning process. In the 1-30000 range, the average scan time spikes to 287.56 seconds, with a standard deviation of 125.59 seconds, indicative of heightened variability and sporadic outliers. Conversely, in the 1-40000 range, while the average scan time remains high at 331.03 seconds, the standard deviation increases further to 196.88 seconds, highlighting substantial variability in scan completion times. In both cases, median scan times align closely with the 75th percentile, suggesting a relatively symmetrical distribution of scan times. Finally, scan times peak dramatically in the expansive 1-50000 range, with an average duration of 589.64 seconds and a standard deviation of 221.79 seconds, underscoring the significant computational resources and time required to complete scans within this range. Despite the considerable variability, percentile analysis reveals that 75% of scans were completed within 750.87 seconds, suggesting a consistent proportion of scans completed within a defined timeframe.

Furthermore, Figure 7.12 gives an overview of the average time taken to perform scans in different port ranges. Beginning with smaller ranges, such as the 100 ports, scans were completed swiftly, averaging a mere 0.2 minutes. As the port ranges expanded, the average scan times proportionally increased: 0.55 minutes for the 1000-port range, 1.83 minutes for 10000 ports, and 3.03 minutes for 20000 ports. Further escalation was observed in larger ranges, with scans of 30000, 40000, and 50000 ports averaging 4.42,



(a) Time Distribution to execute a Full Scan



(b) Time distribution of different port ranges

Figure 7.11: Boxplot representing the time distribution of port scanning 100, 1000, 10k, 20k, 30k, 40k, 50k, 65k ports

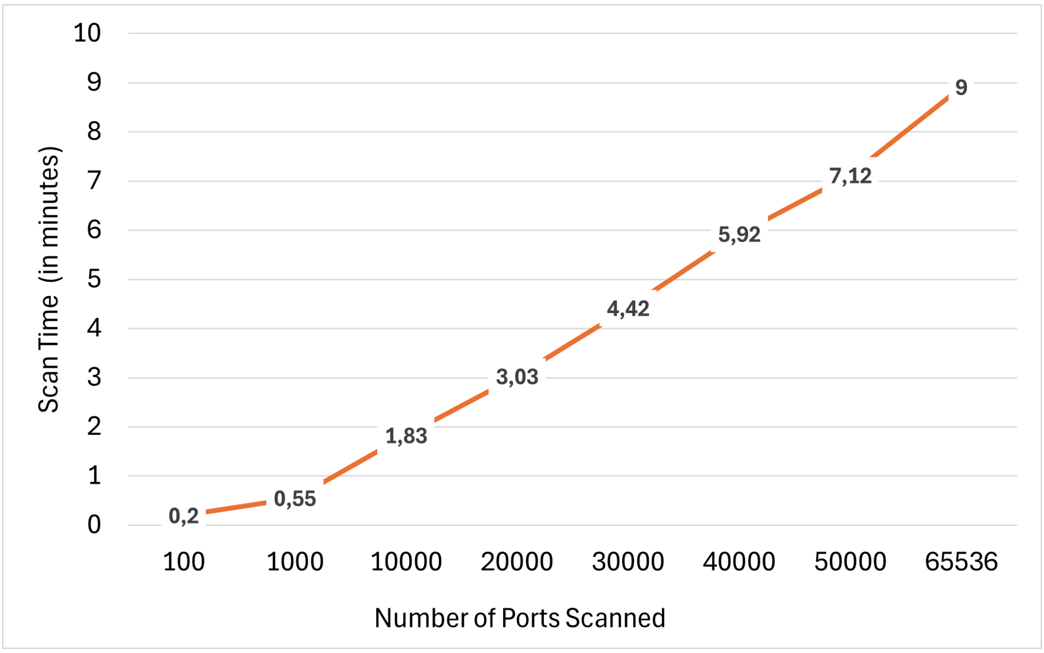


Figure 7.12: Time taken to scan different port ranges

5.92, and 7.12 minutes, respectively. Notably, encompassing the entire port range of 65536 ports required an average time of 9.00 minutes.

These findings collectively illuminate the nuanced dynamics of port scanning, underscoring the trade-offs between scan efficiency, scalability, and computational resources across varying port ranges. However, scaling this approach to a dataset of 100,000 onions would demand roughly 900,000 minutes, equivalent to approximately 1.5 years—a timeframe impractical for most scenarios. These statistics underscore the escalating time demands associated with scanning larger port ranges, emphasizing the need for strategic search-based reduction and optimization planning to balance comprehensive coverage with practical constraints in network security assessments and penetration testing endeavours. By extending the concurrency model to include simultaneous scanning of multiple onions—employing 30 goroutines for onion concurrency—the projected scan duration reduces dramatically, from 1.5 years to just around 20 days. Remarkably, this optimization is achieved using a single tor client instance. Moreover, for even greater acceleration, employing multiple tor client instances can further reduce scan times, enabling expedited analysis and actionable insights.

Port Range	count	mean	std	min	25%	50%	75%	max
1-100	100	18.00	8.14	7.67	12.70	16.54	21.19	63.97
1-1000	100	28.28	22.05	11.32	17.66	23.86	33.29	214.15
1-10000	100	114.16	38.11	36.05	82.58	109.67	137.68	235.90
1-20000	100	167.40	86.11	64.16	118.73	146.68	181.72	709.41
1-30000	100	287.56	125.59	19.05	195.96	265.02	355.45	671.96
1-40000	100	331.03	196.88	47.05	195.61	271.52	427.05	940.13
1-50000	100	589.64	221.79	11.27	431.42	601.10	750.87	971.97

Table 7.2: Statistics of Scan Time per Port Range (time in seconds)

7.2.6. Services

Based on the detection of unique ports, it was observed that multiple ports may correspond to a particular service offered. This observation prompted the grouping of unique ports according to their service type. Table 7.3 presents a comprehensive categorization of protocols based on the type of service they offer, along with the corresponding onion count for each category.

The six services are web, Bitcoin, Remote, Chat, Email, and File Transfer. The web service category encompasses protocols, including HTTPS with 887 onions, alternative HTTP/HTTPS ports with 333 onions, and HTTP with a substantial count of 307,039 onions. In cryptocurrencies, Bitcoin and Monero protocols are represented with 1868 and 305 onions, respectively, while the Lightning Protocol boasts the highest count of 2,470 onions. For remote access, protocols such as Telnet and Gopher, each have 7 and 17 onions, while SSH leads with 1,244 onions. Chat services include the Extensible Messaging and Presence Protocol (XMPP) with 94 onions, Message Send Protocol (MSP) with 182 onions and Internet Relay Chat (IRC) with 74 onions. Email-related protocols encompass a variety of services, with SMTP leading with 87 onions and POP3 following closely with 51 onions. File transfer protocols, including FTP, rsync, and SFTP, collectively account for 45 onions. Other miscellaneous protocols, such as DNS and DHCP, represent 76 onions, highlighting the diverse landscape of services accessible through the Tor network.

Service	Protocol	Onion Count
Web	HTTP	307039
	Alternative HTTP,HTTPS ports	333
	HTTPS	887
Crypto-Asset	Lightning Protocol	2470
	Bitcoin	1868
	Monero	308
Remote	SSH	1244
	UNIX remote management	66
	Telnet	7
	Gopher	17
Chat	Internet Relay Chat (IRC)	74
	Extensible Messaging and Presence Protocol (XMPP)	94
Email	Simple Mail Transfer Protocol (SMTP)	87
	Post Office Protocol (POP3)	51
	Secure Simple Mail Transfer Protocol (SMTPS)	13
	Internet Message Access Protocol (IMAP)	28
	Internet Message Access Protocol Secure (IMAPS)	18
	Post Office Protocol (POP3S)	12
	Network News Transfer Protocol (NNTP)	11
	Ident Protocol	4
File Transfer	rsync	16
	Secure File Transfer Protocol (SFTP)	8
	File Transfer Protocol (FTP)	4
Other	Misc	142
	Gopher Protocol	17
	Xerox Network System (XNS)	10
	Domain Name System (DNS)	9

Table 7.3: Protocols Categorized by Service Type

7.2.7. Temporal Analysis of Services

To further understand the trends in services from 2018 to 2024, services are grouped based on the year they were discovered. This is clearly shown in Figure 7.13. This temporal analysis of onion service adoption reveals significant insights into the dynamic landscape of anonymous communication and online services.

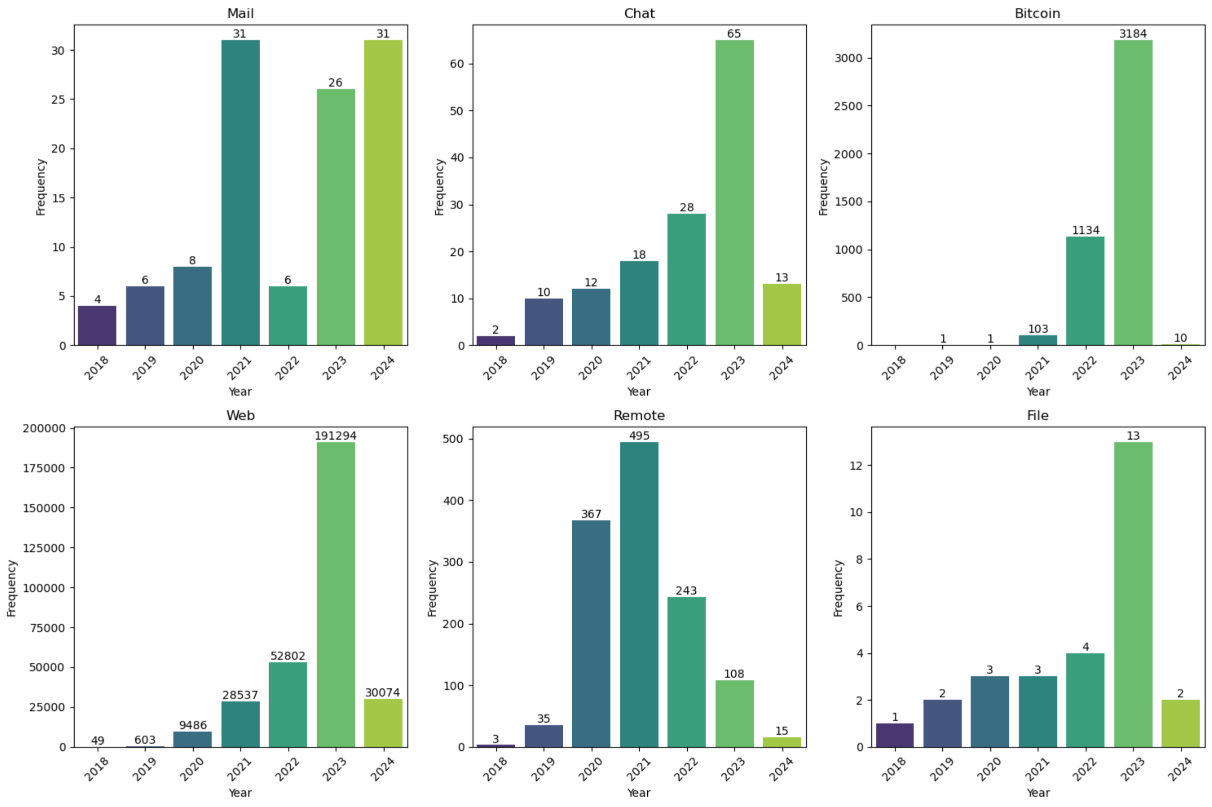


Figure 7.13: Classification of Protocols by Discovery Year

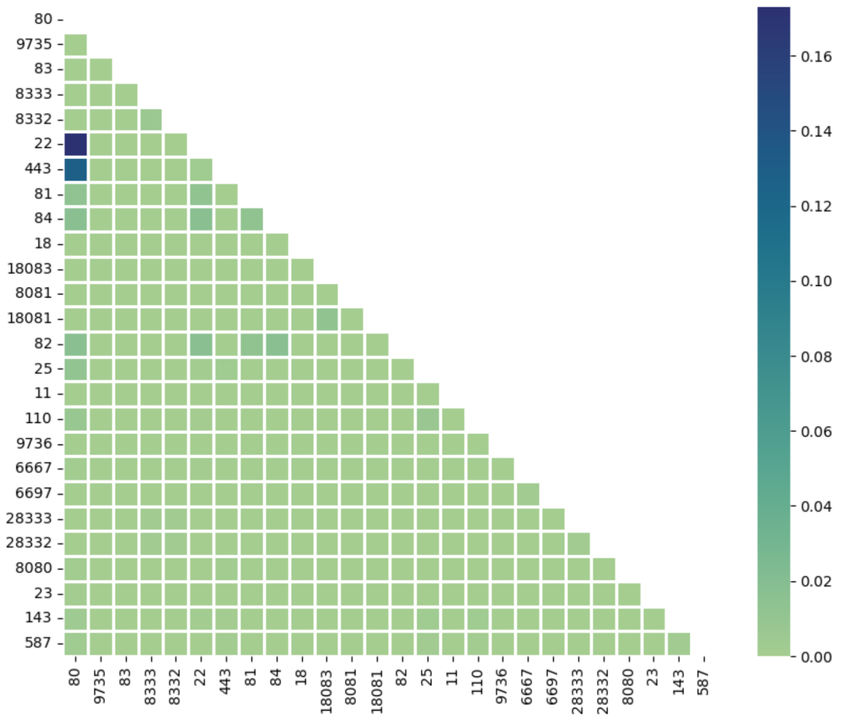
The consistent increase in “Chat” services, from 2 instances in 2018 to 65 in 2023, reflects rising demand for secure messaging platforms driven by concerns over privacy and surveillance. Fluctuations in “Remote” services, from 367 to 495 instances in 2020 and 2021, suggest varying needs for remote access solutions within the Onion ecosystem, possibly influenced by events like the COVID-19 pandemic. The stable yet moderate growth in “Mail” services, ranging from 4 to 31 instances, indicates a steady interest in secure email communication. Similarly, “File” services exhibit fluctuations, signalling varying demand for secure file storage and sharing solutions. Exponential growth in “Web” services, from 49 to 191,294 instances, reflects a significant shift toward onion-based websites, emphasizing anonymity in online browsing. Likewise, the exponential rise in “Bitcoin” services, peaking at 3184 instances, mirrors the surge in cryptocurrency-related activities within the onion network.

Such trends underscore users’ evolving needs and preferences within the Onion ecosystem. Moreover, the consistent upward trajectory observed across the majority of services further emphasizes the critical importance of this thesis in comprehensively understanding the dynamic landscape of anonymous communication and online services.

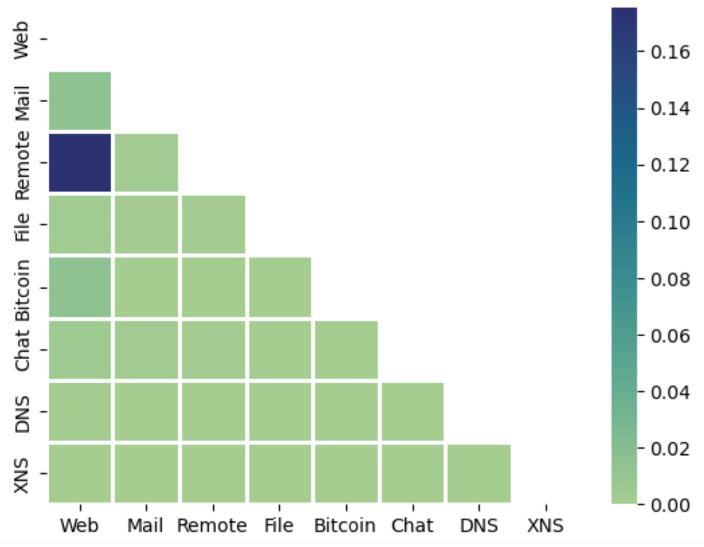
7.2.8. Correlation Analysis: Individual Ports vs. Service-Based Groupings

A correlation matrix, shown in Figure 7.14, was generated to explore the relationships between pairs of ports and pairs of services. This matrix visually illustrates the strength and direction of these correlations. To compute this correlation matrix, onion services exclusively offering web service (limited to ports 80, 443, or both) were discarded. Instead, we concentrated on the remaining data comprising alternative ports, totalling 6200 onion services. This focused analysis aims to uncover specialized use cases within the Tor network.

Examining the correlation matrix of the top 25 frequent ports shown in Figure 7.14a, we observe that Port 80 strongly correlates with port 22 (SSH), as many web servers enable SSH for remote management. Port 80 also shows some correlation between ports 443, 81, 84, and 82 (alternative http/https ports) and with ports 25 and 110 (mail related). Furthermore, a weak correlation is seen between ports 18081



(a) Correlation matrix of top 25 frequent ports



(b) Correlation matrix of service-based groupings

Figure 7.14: Correlation Analysis

and 18083 (Monero). In the correlation matrix of service-based groupings, as shown in Figure **??**, we observe that certain pairs of services show statistically significant correlations. “Web” and “Remote” services indicate the highest positive correlation of 0.16. This suggests that as the frequency of “Web” services increases over time, there is a slight inclination for the frequency of “Remote” services also to increase. However, the relationship is not particularly strong. Similarly, weak positive correlations are evident between “Web” and “Mail” services, as well as between “Web” and “Bitcoin” services.

While some correlations are statistically significant, most correlations in the provided matrix are close to zero, indicating weak or negligible relationships between the frequencies of different onion services. This implies that the adoption and frequency of one type of onion service are generally not strongly influenced by the adoption and frequency of other types of services.

7.3. Optimized Port Scanning Methodology

The optimized port scanning methodology has three phases, as shown in Figure 7.15. Firstly, we scan for open ports for all the onion services in the dataset and then start a thread that revisits the onion services that are online, offline, and in T.U. based on the heuristics defined. There are two key subparts to the final phase, where we perform a monthly comprehensive analysis of onion services. Firstly, we monitor the onion services under different statuses to update the heuristics and when to revisit them. Secondly, we perform a full scan on random samples of onion services to update the list of unique ports observed so far among the onion services. These steps are detailed in the following subsections, and Algorithm 4 presents the pseudocode for the port scanning methodology.

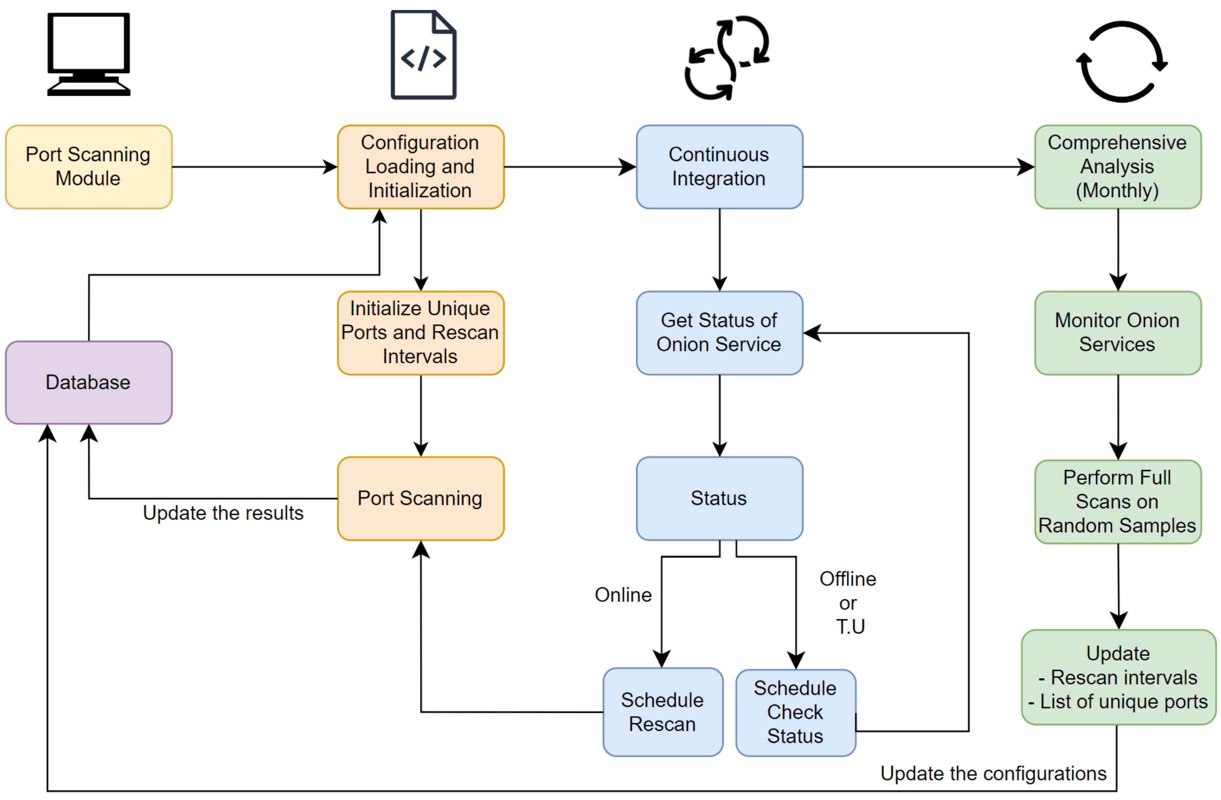

Port Scanning Module
Configuration Loading and Initialization
Continuous Integration
Comprehensive Analysis (Monthly)
Initialize Unique Ports and Rescan Intervals
Database
Get Status of Onion Service
Monitor Onion Services
Port Scanning
Update the results
Status
Perform Full Scans on Random Samples
Online
Offline or T.U
Schedule Rescan
Schedule Check Status
Update - Rescan intervals - List of unique ports
Update the configurations

Figure 7.15: Port Scanning Methodology

7.3.1. Port Scanning

Considering the time to conduct a full scan, we make a search-based reduction to check the open ports on onion services. Algorithm 4 presents the optimised port scanning methodology. It begins with an input text file containing onion services, along with optional parameters such as port range, Tor proxy address, and configuration settings. The desired output is the storage of scan results in a database. The algorithm initializes by loading configuration parameters from the config file, including unique ports, maximum concurrent scans, and rescan intervals. These settings play a crucial role in optimizing the scanning process.

‘unique_ports‘ variable stores a list of unique ports observed during the scanning process. It is initialized with the unique ports loaded from the configuration file and is updated iteratively as new open ports are detected during scanning. Currently through the research, 196 unique ports are detected through port scanning. Maintaining a list of unique ports helps optimize the scanning process by focusing on previously unobserved ports and avoiding redundant scans. ‘rescan_intervals‘ variable stores the intervals at which onion services are scheduled for rescanning based on their status. It is a configuration parameter that defines the frequency of rescans for services that are T.U, offline, or online. By scheduling periodic rescans, the algorithm can maintain an up-to-date view of the status of onion services and adapt dynamically to changes in their availability.

Three main tasks are performed during the scan: 1. Checking onion Service Status: It checks the status of the onion, categorizing it as Online, Offline, or T.U To check the status, an HTTP request is sent on any one port number out of 1-65536, and depending on the response observable, i.e., error messages, we can determine the status as explained in 5.4.1, 2. Scanning Online Services: If the service is online, the function scans the specified port range or the list of unique ports, depending on the configuration. A full scan is triggered if no open ports are detected or more than two ports are open within the given port range. The results are stored in the database following the format outlined in the architecture’s data storage section, 3. Update Unique Ports List: The list of unique ports uses the update_unique_ports function. The update_unique_ports function iterates over the open ports detected during the port scan. It checks each open port detected on that particular onion service against the list of unique ports previously detected and adds any new ports observed to this list. The updated list is then saved to the configuration file for future reference.

7.3.2. Continuous Integration

The continuous integration function is a pivotal component of the port scanning methodology, designed to ensure the accuracy and timeliness of the database containing information about onion services. It operates within an infinite loop, perpetually monitoring the status of onion services and conducting rescans to keep the database up-to-date. At the start of each iteration, the function retrieves the current time, enabling precise scheduling and tracking of rescans.

For each onion service stored in the database, the function retrieves its current status using the get_current_status function. The status can be one of three: T.U, offline, or online. Depending on the status of the onion service, the continuous integration function takes appropriate action. Onion services flagged as ’T.U’ (T.U) undergo thorough monitoring at designated intervals. Specifically, these services are checked at three checkpoints from when they are marked as T.U: firstly, after 50 minutes, followed by scans at 4-hour and 12-hour intervals, as detailed in Section 7.1.2. Offline services are subject to rescans every 24 hours (because of the onion service descriptor update mechanism), while online services are re-evaluated every 18 hours (because it takes 15 hours to scan all the online services currently present in the DWM).

For online services, the function initiates a port scan using the scan_ports function to check for any changes in the open ports since the last scan. If changes are detected, indicating potential new ports or closures, a full scan is performed to find all open ports of the onion service. After each scan, the function logs the scan results, including the status and any changes in the open ports in the database. This logging mechanism records the service’s availability and port configuration, facilitating trend analysis and anomaly detection.

Finally, the continuous integration function schedules the next rescan for each onion service based on its current status, ensuring that online services are regularly re-evaluated and any necessary rescans are conducted according to the specified intervals. By implementing the continuous integration function as a separate thread, the port scanning methodology achieves concurrent execution with other processes, maintaining system efficiency and responsiveness while ensuring the integrity of the database.

These periodic scans contribute to the dynamic nature of the onion services in the dark web and ensure that any changes in the service’s availability are promptly detected and monitored over time.

Algorithm 4: Onion Service Port Scanning: Search-based optimization

```
Input: Text file with onion services, Port Range (optional), Tor Proxy Address,
       Config parameters
Output: Scan results stored in the database
Data: config = load_config()

unique_ports = config['unique_ports'] ;
max_concurrent = config['max_concurrent'] ;
rescan_intervals = config['rescan_intervals'] ;

Function update_unique_ports(open_ports):
    foreach port in open_ports do
        if port not in unique_ports then
            Add port to unique_ports;
        end
    end
    Save unique_ports to config file;

Function scan_ports(onion_service, scan_range, tor_proxy, max_concurrent):
    foreach Onion Service do
        Check service status (Online, Offline, T.U);
        if status == Online then
            if Port Range is specified then
                Scan ports within specified range;
            else
                Scan the unique ports;
                if no open port or more than two open ports are detected then
                    Perform full scan;
                end
            end
            Update the database with scan results;
            update_unique_ports(open_ports);
        end
    end

Function continuous_integration():
    while True do
        current_time = get_current_time();
        foreach onion_service in database do
            status = get_current_status(onion_service);
            if status == 'temporarily_unavailable' then
                schedule_rescan(onion_service, rescan_intervals['temporarily_unavailable']);
            end
            else if status == 'offline' then
                schedule_rescan(onion_service, rescan_intervals['offline']);
            end
            else if status == 'online' then
                open_ports = scan_ports(onion_service, unique_ports, tor_proxy, max_concurrent);
                if changes_detected(open_ports) then
                    scan_ports(onion_service, range(1, 65536), tor_proxy, max_concurrent);
                end
                log_scan_results_in_database(onion_service, open_ports);
                schedule_rescan(onion_service, rescan_intervals['online']);
            end
        end
    end

start_thread(continuous_integration);
```

7.3.3. Comprehensive Analysis to update the heuristics

In addition to the continuous integration in the port scanning methodology, a comprehensive monthly scanning method is implemented to ensure thorough coverage of onion services. There are two key subparts to the final phase. Firstly, we monitor the onion services under different statuses to update the heuristics and determine when to revisit them. This involves taking a sample of 10,000 onion services and monitoring them at varying frequencies: every 5 minutes for T.U onions, every 30 minutes for offline onions, and every hour for online onions. This experiment is repeated with 5 different random samples. For each category, such as T.U onions, we find a point in time when all the onions have changed their state at least once. These points become the heuristics for revisiting the onion services, and this process is done for all three categories. Secondly, we perform full scans on random samples of onion services to update the list of unique ports observed among the onion services. This combined approach ensures that our monitoring process is dynamic and adaptive, allowing us to maintain a thorough understanding of the dark web landscape and the open ports of onion services. This enables us to define efficient heuristics for monitoring onion services, optimize the schedule times employed by the DWM, and update the list of unique open ports observed through comprehensive scanning of onion services.

8 Discussion

In this chapter, we reflect on the results obtained from our investigation into the port scanning of onion services on the dark web. We provide a comprehensive analysis of the findings, interpreting their significance and placing them within the context of existing research. Additionally, we discuss the limitations of this research, identifying areas where improvements could have been made. Through this reflection, we aim to highlight the practical implications of our work for cybersecurity and dark web monitoring.

8.1. Reflections

This section presents a detailed interpretation of the current state of onion services, emphasizing the transition from traditional web-based protocols like HTTP/HTTPS to other application layer protocols such as SSH, SMTP, FTP, POP3, IRC, etc. We also discuss the implementation and effectiveness of our optimized port scanning approach, highlighting significant improvements in efficiency and coverage. Lastly, we consider the practical implications of our findings, providing insights into how these advancements can enhance the monitoring and understanding of the dark web.

8.1.1. Interpretation of Onion Service Landscape

We have transitioned from using traditional HTTP/HTTPS-based methods, which determine the status of an onion service based on the port being checked during a request, to any application-based methods, i.e., independent of the port. We now classify the availability of onion services into three status codes: online, offline, and T.U.

Our analysis of response times for these status codes revealed distinct patterns. Services marked as ’online’ displayed faster response times, indicating stable and reliable connections. Conversely, services classified as ’T.U’ or ’offline’ exhibited slower or inconsistent response times, with T.U services showing the longest response times. This indicates that monitoring response times can effectively assess the availability and reliability of onion services.

We propose defining the availability of onion services based on observed response times. For instance, services taking longer than 60 seconds to respond can be classified as T.U. Understanding response times helps prioritize resources for monitoring more stable services, thus enhancing the efficiency of dark web oversight.

Our study found that the majority of onion services had open ports clustered around commonly used service ports, such as HTTP (80), SSH (22) HTTPS (443). This finding aligns with previous research, but our extended scan also uncovered a notable presence of other service ports that had not been as thoroughly documented in earlier studies. Ports that were not as frequently documented in previous studies are lightning protocol (9735, 9736), bitcoin (8332, 28333, 28332), SMTP (25), XMPP (5222, 5269), POP3 (110), etc, indicating a wider range of services than initially anticipated. These results suggest a more diverse service ecosystem within the dark web than previously understood

8.1.2. Optimized Port Scanning Methodology

Implementing our optimized port scanning methodology, which includes regular scans of unique ports observed (196 ports discovered through this research) and monthly full scans, proved effective in maintaining an updated and accurate mapping of onion services. The regular scanning method's efficiency was validated by its ability to quickly detect changes in the commonly observed ports, while the comprehensive monthly full scans provided a deeper understanding of service dynamics by identifying variations in port usage over time. This dual-method approach highlights the importance of combining regular and extensive monitoring to maintain an accurate and up-to-date mapping of onion services. Our findings underscore the necessity for continuous adaptation and improvement in port scanning tools to keep pace with the evolving dark web landscape.

The time distribution analysis of port scanning played a crucial role in optimizing the efficiency of this process. Our findings indicate that a full scan of a single onion service takes an average of 9 minutes. In stark contrast, previous research by [20] reported a scanning duration of 24 hours for 10,000 ports when investigating a single onion service. This demonstrates a substantial improvement in port scanning efficiency.

This optimization reduces the time and resources required for each scan and allows for more frequent and comprehensive monitoring of onion services. Reducing scanning time enables quicker detection of service availability changes, enhancing the responsiveness of monitoring systems. Furthermore, it allows the monitoring efforts to scale, potentially covering a broader range of onion services within the same timeframe.

8.2. Fingerprinting Onion Services

The process of port scanning onion services has unveiled a range of unexplored opportunities for the fingerprinting of these hidden services. Traditionally, the analysis of onion services has focused primarily on web-based interactions, typically limited to standard HTTP (port 80) and HTTPS (port 443) ports. However, our comprehensive port scanning has revealed that many onion services also have non-HTTP-based ports open, significantly broadening the scope for fingerprinting.

One notable observation is the correlation between ports 80 and 22 (SSH) on these onion services. This correlation suggests a potential pattern in the configuration and deployment of onion services, where web services and secure shell access often coexist. This discovery opens avenues for advanced fingerprinting techniques, enabling the identification and grouping of onion services based on their port configurations and usage patterns.

Furthermore, the fact that many onion services share the same user interface (UI) presents another opportunity for fingerprinting. By analyzing the content and cataloguing these UIs, we can develop unique fingerprints for each service. This approach allows us to combine similar onion services, potentially uncovering relationships and connections between seemingly disparate services. This method of UI-based fingerprinting can be particularly effective in identifying cloned or related services that may be part of a larger network or operation.

Banner grabbing, a technique used to gather information about a service by capturing the banner it returns, can also be leveraged in this context. We can collect detailed information about the software and versions running on these onion services by performing banner grabbing on various ports, including non-HTTP ports. This information is invaluable for constructing accurate fingerprints and enhancing our understanding of the underlying infrastructure of the dark web.

Altogether, the insights gained from port scanning and subsequent fingerprinting efforts highlight the potential for a more nuanced and comprehensive analysis of onion services. We can develop richer, more detailed fingerprints by moving beyond a web-based view to consider a wider array of application-level interactions. These fingerprints help in categorizing and grouping onion services and provide a deeper understanding of their operational characteristics and potential interconnections.

8.2.1. Practical Implications

Our research on port scanning in the dark web has several practical implications that can enhance the effectiveness and security of monitoring and understanding onion services. These implications are critical for various stakeholders, including cybersecurity professionals, researchers, law enforcement agencies, and policymakers.

We move from the traditional web-based methods to any application-based method, independent of the port being checked to determine the availability of onion services. This strategy ensures that onion services with non-HTTP-based open ports are not mistakenly categorized as offline, providing a more accurate representation of their availability. The optimized port scanning methodologies developed in this research allow for more frequent and comprehensive monitoring of onion services. By identifying and classifying onion services based on response times and port usage, security professionals can better understand the structure and behaviour of dark web services. This understanding helps anticipate potential threats and vulnerabilities, allowing for more proactive and targeted security measures. The classification of services into 'online', 'offline', and 'T.U' based on response times aids in prioritizing monitoring efforts. Resources can be directed towards more stable services, ensuring efficient use of computational and human resources in maintaining the security and reliability of the network.

Moreover, existing literature indicates that traditional port scanning methods can take up to 24 hours to scan 10,000 ports on a single onion service, and there are no effective tools currently available that can perform this task efficiently. This makes determining the open ports on an onion service a particularly challenging endeavour. Our research addresses this issue by providing a more efficient and accurate method for identifying open ports on onion services. This advancement not only facilitates a better understanding of these services but also equips legal authorities with the necessary information to take further action, such as exploiting vulnerabilities identified through these scans. It is important to note that such activities should be conducted strictly within the bounds of the law, as offensive hacking is illegal for individuals without proper authorization. Our research thus empowers authorized entities to enhance their investigative capabilities while maintaining ethical and legal standards.

This research significantly contributes to the academic body of knowledge on the dark web by comprehensively analysing Tor v3 onion services. Previous studies primarily focused on v2 onion services, leaving a gap in understanding the newer v3 services. By filling this gap, our research offers a more complete and current picture of the Tor network, laying the groundwork for future studies and enabling researchers to build upon a more thorough understanding of its complexities and challenges.

Research on port scanning on the dark web advances the technical field. It has a far-reaching impact on various aspects of society, contributing to a safer and more secure digital environment.

8.3. Limitations

This section outlines the primary limitations encountered during our research on port scanning within the Tor network. Despite our efforts to employ robust methodologies and optimize our scanning processes, several challenges impacted the consistency and efficiency of our results. These limitations include the inherent instability of the Tor network and the constraints posed by our computing resources, such as RAM, which affected our ability to achieve faster scan times through higher concurrency values.

8.3.1. Instability of the Tor Network

Conducting research on port scanning within the Tor network presents several challenges, primarily due to the network's inherent instability. The Tor network's performance can vary dramatically over short periods. Factors such as network congestion, relay node availability, and the performance of individual onion services can lead to significant fluctuations in response times and availability. As a result, the same onion service might appear online at one moment and offline the next, complicating the assessment of onion service stability. The Tor network relies on a global network of volunteer-operated relay nodes, whose availability and performance frequently change. Variations in relay node reliability can affect the routing paths taken by requests, introducing additional unpredictability into port scanning results. Different paths may have different latencies and stabilities, influencing the results of our scans at different times.

To mitigate these challenges, we employed multiple scans at different times and used a combination of regular and full scans to enhance data reliability. Despite these efforts, the unstable behaviour of the Tor network remains a significant limitation, highlighting the need for ongoing methodological improvements. This instability must be considered when interpreting the results of our research and underscores the importance of continuous monitoring and adaptive strategies in studying the Tor network.

8.3.2. Computing Resources

Our research was conducted using the research server provided by the company CFLW Cyber Strategies. While Ganymede offered substantial computational capabilities, we encountered limitations related to available RAM. Higher RAM would have enabled us to use higher concurrency values during port scans, significantly improving scan speed and efficiency. The restricted RAM resources constrained our ability to perform rapid and extensive scans, potentially affecting the comprehensiveness and timeliness of our results. Addressing these limitations in future research by leveraging more advanced computing resources or optimizing memory usage could lead to more efficient and accurate port scanning processes.

9 Conclusion

This concluding chapter summarizes our research's key findings and contributions to port scanning within the dark web. This research provides a detailed analysis of the current state of onion services and introduces optimized scanning methodologies that significantly improve scan efficiency. Additionally, we discuss potential directions for future work, which aim to further refine and expand upon our findings to better address the evolving challenges of the dark web landscape.

9.1. Summary

Our primary objective was to uncover the prevalent protocols used for communication on the Dark Web and to explore the activities beyond traditional web protocols such as HTTP or HTTPS. The main research question is:

RQ: What prevalent protocols are utilized for communication on the Dark Web, and what types of activities occur beyond traditional web protocols such as HTTP or HTTPS?

In this thesis, we conducted an in-depth investigation into the Tor Network Protocols Landscape, focusing on the latest v3 version of Tor onion services. To address the RQ, four SQs were formulated. The first SQ is related to building a tool.

SQ1: How can we develop a specialized port scanning tool tailored specifically for Dark Web environments?

Building upon the framework of the OnionScan tool, we have developed a more efficient and faster port scanning tool, enabling us to scan onion services for open ports. The extended tool can now scan any port range surpassing the default capability limited to the top 10 common ports. In addition to port scanning, the modified tool incorporates functionalities for identifying the availability status of onions, categorizing them as online, offline, or temporarily down. To optimize performance and efficiency, the tool supports the utilization of goroutines. Moreover, integrating MySQL database functionality seamlessly complements the scanning tool, facilitating the systematic storage of scan results. Now that the tool is ready, the second SQ was to investigate the onion services landscape:

SQ2: What are the key characteristics and dynamics of the onion services landscape within the Dark Web?

Our study of the Tor onion service mechanism has yielded several key insights into the availability status and characteristics of onions within the Dark Web ecosystem. By categorizing onion services into three distinct groups—online, offline, and T.U, we have advanced beyond the conventional web-based method of status determination, which was limited to checking ports 80 and 443. Instead, we employ an application-based approach that is independent of specific ports. Of the three categories of onion services, T.U ones were a relatively small proportion. The T.U onion services were monitored over a 2-day period at 5-minute intervals. This experiment aimed to define heuristics for revisiting T.U onion services. Based on the analysis, we propose heuristics for revisiting these onions at specific intervals, such as 50 minutes, 4 hours, and 12 hours, until each T.U. onion has experienced at least one status change.

SQ3: How do port scanning activities contribute to uncovering the prevalent protocols within the Dark Web ecosystem

Our research has revealed the existence of onions that are active but not open on web-based ports 80 and 443, challenging the conventional definition of online status. To address this, we have defined availability status independent of the port being checked, focusing on the type of response received when an HTTP connection is made. Our analysis has shown that the majority of online onions have port 80 open, with only a small percentage having ports that are not open on 80. We have detected 196 unique ports across the scanned onions, with the most prevalent ports being associated with lightning, bitcoin, SSH, SMTP, and POP3 services.

Furthermore, we have grouped the open ports based on the default services associated with them, resulting in six distinct service groups: Web, Bitcoin, Remote, Chat, Email, and File Transfer. Our investigation has revealed the prevalence of remote, chat, and email services within the Tor network, alongside web and bitcoin services. Additionally, our temporal analysis has shown that onions discovered by the Dark Web Monitor between 2018 and 2022 remain active, highlighting the persistence of onion services.

SQ4: How can we optimize traditional port-scanning methods to suit the challenges of Dark Web networks better suit the challenges of Dark Web networks better?

Notably, our research has demonstrated the efficiency of our proposed optimized port scanning methodology. Recognizing the impracticality of conducting a full scan on all onion services, we have developed a methodology that prioritizes scanning based on predefined criteria, thereby reducing scan times while maintaining comprehensive coverage. With a full scan of one onion service taking approximately 8 minutes—the fastest reported in the literature to date—our methodology offers a practical solution to the challenges of port scanning within the Tor network.

Our port-scanning methodology has three phases. The first phase included port scanning. The second phase is continuous integration, where onion services are revisited, perpetually monitoring the status of onion services and conducting rescans to keep the database up-to-date. In the third phase we perform a monthly comprehensive analysis of onion services. The objective is to monitor the onion services under different statuses to update the heuristics on when to revisit them and perform a full scan on random samples of onion services to update the list of unique ports observed so far among the onion services.

9.2. Future Work

Building upon the contributions outlined in this thesis, several avenues for future research and development emerge, further enhancing our understanding of the Tor Network Protocols Landscape and improving the efficiency of port-scanning methodologies within the Tor network.

1. **Fingerprinting of Services** While the current research focuses on detecting open ports within Tor onion services, future work should delve into service fingerprinting. Confirming the services running on the detected open ports is crucial for several reasons. Firstly, it allows researchers to validate the findings and ensure the accuracy of the port scanning results. The ports open can run any service; for example, an onion service with port 22 open could be configured to run a mail service instead of the default SSH service. Hence, by definitively identifying the services associated with the open ports, researchers can eliminate false positives and better interpret the significance of the detected ports.
2. **Testing on Higher Configuration Processors** The tool used for port scanning in this research operates on a research server with 64 GB RAM, shared among multiple users for research purposes. Future work should involve testing the tool on higher configuration processors to enhance the scanning efficiency and accommodate larger-scale scanning operations. Researchers can optimize scanning speed and scalability by utilising more powerful hardware resources, facilitating a more extensive and in-depth analysis of the Tor network.
3. **Scanning larger sample of onion services** Besides enhancing scanning efficiency, future research efforts should focus on expanding the scope of scanning operations. Specifically, researchers should aim to scan a larger number of onion services available from the Dark Web Monitor. By increasing the sample size and diversity of scanned onion services, researchers can gain further insights into the prevalence and distribution of various protocols within the Tor network. This expanded dataset will enable a more comprehensive analysis of Tor network dynamics and facilitate the identification of emerging trends and patterns.

4. **Public Web connection** Investigating the linkage between onion services and public web services is a promising direction for future research. This includes analyzing mirror sites, crossover domains, and the extent of data and service replication between the dark web and the public internet. Understanding these linkages can help identify the reach and influence of dark web services on the broader web ecosystem, providing valuable insights into the interaction and interdependence between these two distinct parts of the internet.

References

[1] The Tor Project. *Tor Project - History*. https://www.torproject.org/about/history/.

[2] Keith Kirkpatrick. "Financing the Dark Web". In: *Commun. ACM* 60.3 (Feb. 2017). https://doi.org/10.1145/3037386, pp. 21–22. DOI: 10.1145/3037386.

[3] *Tor Metrics*. https://metrics.torproject.org/.

[4] Tanja Miloshevska. "Dark Web as a Contemporary Challenge to Cyber Security". In: *Kriminalističke teme* 5 (Oct. 2019), pp. 117–128. URL: https://krimteme.fkn.unsa.ba/index.php/kt/article/view/219.

[5] Douglas Everson and Long Cheng. "A Survey on Network Attack Surface Mapping". In: *Digital Threats: Research and Practice* (Jan. 2024). DOI: 10.1145/3640019.

[6] Saiba Nazah, Shamsul Huda, Jemal Abawajy, and Mohammad Mehedi Hassan. "Evolution of Dark Web Threat Analysis and Detection: A Systematic Approach". In: *IEEE Access* 8 (2020), pp. 171796–171819. DOI: 10.1109/ACCESS.2020.3024198.

[7] D. Seaton. *Understanding The Dark Web. 2020.* https://cyberauditteam.com/blog/identify/understanding-the-dark-web.

[8] Center for Internet Security. *Election Security Spotlight – The Surface Web, Dark Web, and Deep Web*. https://www.cisecurity.org/insights/spotlight/cybersecurity-spotlight-the-surface-web-dark-web-and-deep-web.

[9] Shubhdeep Kaur and Sukhchandan Randhawa. "Dark web: A web of crimes". In: *Wireless Personal Communications* 112 (2020). https://doi.org/10.1007/s11277-020-07143-2, pp. 2131–2158.

[10] Bryan Monk, Julianna Mitchell, Richard Frank, and Garth Davies. "Uncovering Tor: An Examination of the Network Structure". In: *Security and Communication Networks* 2018 (May 2018), pp. 1–12. DOI: 10.1155/2018/4231326.

[11] VPN Unlimited. *What is L2TP VPN Protocol?* https://www.vpnunlimited.com/help/vpn-protocols/l2tp-protocol.

[12] Roger Dingledine, Nick Mathewson, and Paul Syverson. "Tor: The Second-Generation Onion Router". In: *13th USENIX Security Symposium (USENIX Security 04)*. San Diego, CA: USENIX Association, Aug. 2004. URL: https://www.usenix.org/conference/13th-usenix-security-symposium/tor-second-generation-onion-router.

[13] H. Neal. *SVG Diagram of the "Onion Routing"Principle. 2008.* https://commons.wikimedia.org/wiki/File:Onion_diagram.svg.

[14] Chunmian Wang, Junzhou Luo, Zhen Ling, Lan Luo, and Xinwen Fu. "A comprehensive and long-term evaluation of tor v3 onion services". In: *Proceedings of the 42nd IEEE International Conference on Computer Communications (INFOCOM)*. IEEE. 2023.

[15] EC-Council. *What is Penetration Testing*. https://www.eccouncil.org/cybersecurity-exchange/penetration-testing/what-is-penetration-testing.

[16] EC Council Global Services. *Vulnerability Assessment and Penetration Testing*. https://egs.eccouncil.org/services/vulnerability-assessment-and-penetration-testing.

[17] Marco De Vivo, Eddy Carrasco, Germinal Isern, and Gabriela O De Vivo. "A review of port scanning techniques". In: *ACM SIGCOMM Computer Communication Review* 29.2 (1999), pp. 41–48.

[18] S Bauer. "Evaluation of Port Scan and Port Scan Detection Tools". In: *Bachelor's Thesis* (2015).

[19] Nmap. `https://nmap.org/book/port-scanning.html`.

[20] Fatin Hazirah Roslan. “A Comparative Performance of Port Scanning Techniques”. In: *Journal of Soft Computing and Data Mining* 4.2 (2023), pp. 43–51.

[21] Nazar El-Nazeer and Kevin Daimi. “Evaluation of Network Port Scanning Tools”. In: *Proceedings of the International Conference on Security and Management (SAM)*. The Steering Committee of The World Congress in Computer Science, Computer … 2011, p. 1.

[22] Alex Biryukov, Ivan Pustogarov, Fabrice Thill, and Ralf-Philipp Weinmann. “Content and Popularity Analysis of Tor Hidden Services”. In: *2014 IEEE 34th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. 2014, pp. 188–193. DOI: `10.1109/ICDCSW.2014.20`.

[23] Gareth Owenson, Sarah Cortes, and Andrew Lewman. “The darknet’s smaller than we thought: The life cycle of Tor Hidden Services”. In: *Digital Investigation* 27 (2018), pp. 17–22.

[24] Martin Steinebach, Marcel Schäfer, Alexander Karakuz, Katharina Brandl, and York Yannikos. “Detection and analysis of tor onion services”. In: *Proceedings of the 14th International Conference on Availability, Reliability and Security*. 2019, pp. 1–10.

[25] Alejandro Buitrago López, Javier Pastor Galindo, and Félix Gómez Mármol. “Exploring the availability, protocols and advertising of Tor v3 domains”. In: *2023 JNIC Cybersecurity Conference (JNIC)*. IEEE. 2023, pp. 1–8.

[26] Alejandro Buitrago López, Javier Pastor-Galindo, and Félix Gómez Mármol. “Updated exploration of the Tor network: advertising, availability and protocols of onion services”. In: *Wireless Networks* (2024). DOI: `10.1007/s11276-024-03679-4`.

[27] CFLW Cyber Strategies. *Dark Web Monitor*. `https://cflw.com/academy/`.

[28] Internet Assigned Numbers Authority (IANA). `https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml`.

[29] TOR Project. *Tor’s extensions to the SOCKS protocol*. `https://spec.torproject.org/socks-extensions#tors-extensions-to-the-socks-protocol`.

[30] OnionScan. *Open Source Tool*. `https://github.com/s-rah/onionscan`.

A
Appendix

A.1. Open Ports in Onion Services

This appendix categorizes Onion services based on the presence or absence of port 80. The open ports identified during our research are listed under each category. A total of 196 unique ports were observed, with 147 ports in services where port 80 is open and 92 ports in services where port 80 is not open. Notably, 43 ports are common to both groups.

A.1.1. Onion Services with Port 80 Open

The following ports were observed in Onion services, which have port 80 open. This category highlights the variety of services accessible through the standard HTTP port alongside other ports.

The open ports are 11, 18, 21, 22, 23, 25, 52, 53, 67, 70, 79, 80, 81, 82, 83, 84, 110, 113, 119, 143, 195, 222, 235, 300, 433, 443, 444, 465, 513, 587, 701, 873, 993, 995, 1123, 1222, 1234, 1337, 1716, 1813, 1935, 1965, 1990, 2083, 2202, 2221, 2222, 2345, 2525, 2628, 2880, 2999, 3000, 3012, 3333, 4000, 4190, 4242, 4422, 4569, 5000, 5060, 5061, 5201, 5202, 5222, 5223, 5269, 5270, 5280, 5281, 5400, 5443, 5555, 6667, 6668, 6697, 6698, 6699, 7007, 7777, 7778, 7779, 7890, 7922, 8000, 8001, 8002, 8003, 8004, 8005, 8006, 8080, 8081, 8083, 8085, 8088, 8090, 8333, 8443, 8444, 8446, 8448, 8888, 8889, 9000, 9418, 9443, 9735, 9877, 9911, 9980, 10022, 11181, 11371, 13122, 16000, 17750, 18080, 18081, 18082, 18083, 18089, 19332, 19734, 19950, 22548, 37888, 38081, 40322, 40392, 42001, 44203, 47889, 47922, 48782, 50001, 50002, 50003, 50004, 54221, 54231, 56886, 60001, 60002, 60004, 64738

A.1.2. Onion Services without Port 80 Open

The following ports were observed in Onion services that do not have port 80 open. This category demonstrates the variety of services that operate independently of the standard HTTP port.

The open ports are 22, 23, 25, 53, 70, 110, 119, 143, 194, 443, 465, 587, 853, 993, 994, 995, 1337, 1990, 5222, 5223, 5269, 5280, 5443, 5555, 6660, 6661, 6662, 6663, 6664, 6665, 6666, 6667, 6668, 6669, 6679, 6690, 6691, 6692, 6693, 6694, 6695, 6696, 6697, 6698, 6699, 7000, 7070, 7777, 8067, 8080, 8081, 8332, 8333, 8334, 8443, 8667, 9000, 9236, 9418, 9735, 9736, 9737, 9788, 9889, 9911, 9999, 10009, 10010, 11236, 16667, 18080, 18081, 18083, 18084, 18085, 18089, 18090, 18180, 28080, 28081, 28083, 28089, 28332, 28333, 30029, 34568, 38081, 38089, 45871, 50001, 50002, 64738

A.1.3. Common Ports Between Both Groups

The following ports are common to both Onion services with port 80 open and those without port 80 open. These ports indicate services broadly used across different configurations of Onion services.

The ports common between both groups are 22, 23, 25, 53, 70, 110, 119, 143, 443, 465, 587, 993, 995, 1337, 1990, 5222, 5223, 5269, 5280, 5443, 5555, 6667, 6668, 6697, 6698, 6699, 7777, 8080, 8081, 8333, 8443, 9000, 9418, 9735, 9911, 18080, 18081, 18083, 18089, 38081, 50001, 50002, 64738

A.2. Grouping ports into services based on IANA port mapping

The following table represents all the 196 unique ports detected in the dark web grouped according to the services following the IANA mapping of ports with their respective services.

Port	**Service**
11	systat (port the status of a computer’s system)
18	Message send protocol (messaging)
21	FTP
22	SSH
23	Telnet
25	SMTP
52	XNS Time protocol
53	DNS
67	Bootstrap Protocol Server
70	Gopher
79	Finger
80	HTTP
81	Alternative HTTP
82	Xfer
83	MIT ML Device
84	Common Trace Facility
110	Post office protocol
113	Authentication Service
119	Network News Transfer Protocol
143	Internet Message Access Protocol
194	IRC
195	DNSIX Network Level Module Audit
222	Berkley rshd with SPX auth
235	Reserved
300	Unassigned
433	Network News Transfer Protocol for transit servers
443	HTTPS
444	Simple Network Paging Protocol
465	Message Submission over TLS Protocol
513	Remote login telnet
587	Message Submission
701	Link Management Protocol
853	DNS over TLS
873	rsync
993	IMAP over TLS
994	Reserved
995	POP3 over TLS
1123	Murray
1222	SNI R&D network
1234	Infoseek Search Agent
1337	menandmice DNS
1716	xmsg
1813	Radius Accounting
1935	Macromedia Flash Communications Server MX

1965	Tivoli NPM
1990	cisco STUN Priority 1
2083	Secure Radius Service
2202	Int. Multimedia Teleconferencing Consortium
2221	EtherNet/IP over TLS
2222	EtherNet/IP I/O
2345	DBM
2525	MS-V-Worlds
2628	DICT
2880	Synapse Transport
2999	RemoteWare
3000	RemoteWare Client
3012	Trusted Web Client
3333	DEC Notes
4000	Terabase
4190	ManageSieve Protocol
4242	VRML Multi User Systems
4422	TSEP Installation Service Protocol
4569	Inter-Asterik Exchange
5000	complex-main
5060	SIP
5061	SIP-TLS
5201	TARGUS GetData 1
5202	TARGUS GetData 2
5222	XMPP client-connection
5223	HP VM Group Management
5269	XMPP Server Connection
5270	Cartographer XMP
5280	Bidirectional Streams Over Synchronous HTTP
5281	Undo License Manager
5400	Excerpt search
5443	Pearson HTTTS
5555	Personal Agent
6660-6664	Unassigned
6665-6669	IRCU
6679	OSORNO Automation
6690	CLEVERDetect Message Service
6691-6695	Unassigned
6696	Babel Routing Protocol
6697	IRC via TLS
6698	Reserved
6699	Babel Routing Protocol over DTLS
7000	File Server itself
7007	basic overseer process
7070	ARCP

7777	cbt
7778	Interwise
7779	VSTAT
7890	Unassigned
7922	Unassigned
8000	IRDMI
8001	VCOM Tunnel
8002	Teradata ORDBMS
8003	Mulberry Connect Reporting Service
8004	Opensource Evolv Enterprise Platform P2P Network Node Connection Protocol
8005	MXI Generation II for z/OS
8006	World Programming analytics
8067	Infinidat async replication
8080	HTTP Alternate
8081	Sun Proxy Admin Service
8083	Utilistor(Server)
8085	Unassigned
8088	Radan HTTP
8090	Vehicle to station messaging
8332-8334	Bitcoin
8443	PCsync HTTPS
8444	PCsync HTTP
8446	Unassigned
8448	Matrix Federation Protocol
8667	Unassigned
8888	NewsEDGE server UDP (UDP 1)
8889	Desktop Data TCP 1
9000	CSlistener
9236	Unassigned
9418	git pack transfer service
9443	WSO2 Tungsten HTTPS
9735-9737	Lightning Network
9788	Unassigned
9877	The X.510 wrapper protocol
9889	Port for Cable network related data proxy or repeater
9911	SYPECom Transport Protocol
9980	Unassigned
9999	distinct
10009	Systemwalker Desktop Patrol
10010	ooRexx rxapi services
10022	Unassigned
11181	Unassigned
11236	Unassigned
11371	OpenPGP HTTP Keyserver

13122	Unassigned
16000	Administration Server Access
16667	Unassigned
17750	Unassigned
18080-18090	Monero
18180	Unassigned
19332	Unassigned
19734	Unassigned
19950	Unassigned
22548	Unassigned
28080	thor/server - ML engine
28081	Unassigned
28083	Unassigned
28089	Unassigned
28332-28333	Unassigned
30029	Unassigned
34568	Unassigned
37888	Unassigned
38081	Unassigned
38089	Unassigned
40322	Unassigned
40392	Unassigned
42001	Unassigned
44203	Unassigned
45871	Unassigned
47889	Unassigned
47922	Unassigned
48782	Unassigned
50001-50004	Unassigned
54221	Unassigned
54231	Unassigned
56886	Unassigned
60001-60002	Unassigned
60004	Unassigned
64738	Murmur Server

Table A.1: IANA Service Name and Transport Protocol Port Number Registry