Abusing Privacy Infrastructures: Case Study of Tor

Amirali Sanatinia
College of Computer and Information Science
Northeastern University

A thesis submitted for the degree of
Doctor of Philosophy in Computer Science
PhD 2018
THESIS TITLE: Abusing Privacy Infrastructures: Case Study of Tor

AUTHOR: Amirali Sanatinia

Ph.D. Thesis Approved to complete all degree requirements for the Ph.D. Degree in Computer Science.

Guevara Noubir 4/4/2018
Thesis Advisor

Agnes Chan 4/4/2018
Thesis Reader

Alina Oprea 4/4/2018
Thesis Reader

Erik-Oliver Blass 4/4/2018
Thesis Reader

Aziz Mohaisen 4/4/2018
Thesis Reader

GRADUATE SCHOOL APPROVAL:

4/9/18
Director, Graduate School

COPY RECEIVED IN GRADUATE SCHOOL OFFICE:

4/9/18
Recipient's Signature

Distribution: Once completed, this form should be scanned and attached to the front of the electronic dissertation document (page 1). An electronic version of the document can then be uploaded to the Northeastern University-UMI website.
Committee

Prof. Guevara Noubir (Advisor), Northeastern University

Prof. Agnes Chan, Northeastern University

Prof. Alina Oprea, Northeastern University

Dr. Erik-Oliver Blass (External), Airbus

Prof. Aziz Mohaisen (External), University of Central Florida
Abstract

In the last two decades, advances in privacy-enhancing technologies, including cryptographic mechanisms, standardized security protocols, and infrastructures, have significantly improved the privacy of users. Tor, a byproduct of those primitives, emerged as a practical solution to protecting the privacy of citizens against censorship and tracking. At the same time, Tor’s success encouraged illegal activities, including sophisticated botnets, ransomware, and a marketplace for drugs and contraband. The goal of this thesis is to provide mechanisms that will make detection of abusers of this valuable infrastructure possible.

In this thesis, we investigate the abuse of privacy infrastructures from three different perspectives: (1) we look at the next generation of resilient botnets that rely on Tor for their malicious activity; (2) we expose malicious snooping actors inside the Tor network that are integral to the functioning of the hidden services and the dark web; and (3) we propose a privacy preserving data collection and analysis framework to study the lifespan of hidden services.

In the first part of this thesis, we will preemptively investigate the design and mitigation of such botnets, (i.e., OnionBots) that can achieve a low diameter and a low degree and be robust to partitioning under node deletions. Botnets rose to be a major tool for cyber-crime and their developers proved to be highly resourceful. We contend that the next waves of botnets will extensively attempt to subvert privacy infrastructures and cryptographic mechanisms.

Tor’s security relies on the fact that a substantial number of its nodes do not misbehave. In the second part, we expose a category of misbehaving Tor relays, Hidden Service Directories (HSDirs), that are integral to the functioning of the hidden services and the dark web. The HSDirs act as the DNS directory for the dark web. Because of their nature, detecting their malicious intent and behavior is much harder. We introduce the concept of Honey Onions (HOnions), a framework to detect misbehaving HSDirs.

Very little is known about the lifespan of hidden services. Gaining knowledge and insight about their lifespan provides manifold benefits, such as the detection of malicious and benign domains. However, to avoid disrupting Tor and its security and privacy services, such study needs to be carried out in a privacy-preserving manner. The distributed nature of Tor and hidden services makes such study non-trivial and introduces challenges that need to be addressed. In the last part of this thesis, we devise protocols and algorithms to draw conclusions about hidden services’ dynamics, based on the data collected from the network, while protecting the privacy and security properties of the Tor infrastructure.
To my parents...
I would like to express my sincerest gratitude to my PhD advisor and mentor, Professor Guevara Noubir. I learned a lot from him, both about scientific research and life in general. I also very much enjoyed our discussions and conversations throughout these years. I am forever grateful for his kindness, mentorship, and support.

I am grateful to my committee members, Professor Agnes Chan, Professor Alina Oprea, Dr. Erik-Oliver Blass, and Professor Aziz Mohaisen, for accepting to be on my PhD committee and their support. I appreciate their valuable time and feedback which helped me to improve my research.

I am thankful to Professor Engin Kirda and Professor Rajmohan Rajaraman for their help and support during my years at Northeastern University. A special thanks goes to my teammates, colleagues, and friends at CCIS who made this journey joyful.

I would like to thank my sister and brother, Rahil Sanatinia and Reza Sanatinia, who have unconditionally helped and supported me.

I would like to dedicate my thesis to my dear parents, Aliakbar Sanatinia and Zohre Mohimi, and thank them for everything that I have achieved. I am and will always be grateful to them for their unconditional love and believing in me; without them none of this would have been possible...

Amirali Sanatinia  
Boston, April 2018
Contents

1 Introduction
   1.1 OnionBots: Next Generation Botnets ........................................... 2
   1.2 HOnions: Exposing Snooping HSDirs ............................................ 3
   1.3 Privacy Preserving Study of Hidden Services Lifespan .................... 5
   1.4 Outline ................................................................. 5

2 Privacy Infrastructure: Tor ......................................................... 7
   2.1 Tor Hidden Services ........................................................... 7
   2.2 Tor and Hidden Services Performance ....................................... 11
   2.3 Related Work ............................................................... 14
   2.4 Next Generation Hidden Services .......................................... 15
      2.4.1 Shared Random Value .................................................... 15
      2.4.2 Encryption and Signing Keys .......................................... 16

3 OnionBots: Next Generation Botnets ............................................. 19
   3.1 Related Work ............................................................... 21
   3.2 OnionBot: a Cryptographic P2P Botnet ...................................... 25
      3.2.1 Overview ............................................................. 25
      3.2.2 Bootstrap ............................................................. 26
      3.2.3 Maintenance of the OnionBot Communication Graph ............... 28
      3.2.4 Command and Control Communication ................................ 30
      3.2.5 Operation ............................................................. 32
   3.3 OnionBots Evaluation .......................................................... 34
      3.3.1 Mapping OnionBot ...................................................... 34
      3.3.2 OnionBot Network Resilience ....................................... 36
   3.4 Mitigation of Basic OnionBots ............................................... 40
      3.4.1 Targeting OnionBots Through Privacy Infrastructures ............. 42
      3.4.2 Sybil Onion Attack Protocol (SOAP) ................................ 42
   3.5 Beyond Basic OnionBots ....................................................... 44
      3.5.1 Overcoming SOAP ...................................................... 45
# List of Figures

1.1 Tor’s advertised vs. utilized bandwidth. ........................................ 3
1.2 Number of unique hidden services. ............................................ 4

2.1 Onion Routing, as it is used in Tor to establish circuits. .............. 8
2.2 Tor hidden service architecture and connection setup. ............... 9
2.3 Fingerprints circle or Hidden Service Directories (HSDir) and placement of a hidden service descriptor. ................................. 10
2.4 Tor and hidden services’ performance measurement setup. We use different scenarios to measure the performance of Tor OR connecting to a normal HTTP server and also hidden services. ............ 12
2.5 Tor and hidden services’ performance. Increasing the number of client connections on the same Tor OR, decreases performance. However, for the hidden services, each Tor OR can optimally support multiple hidden services. ................................. 13

3.1 Comparison of Single-flux and Double-flux networks, from The Honeynet Project [34]. ................................................................. 23
3.2 Node removal and self repairing process in a 3-regular graph with 12 nodes. The dashed red lines, indicate the newly established links between the nodes. ................................................................. 28
3.3 The average closeness centrality (3.3a, 3.3b) and degree centrality (3.3c, 3.3d) of nodes in a k-regular graph, (k = 5, 10, 15) with 5000 nodes after 30% node deletions, with and without pruning. ............ 35
3.4 Graphs depicting the number of connected components, and average degree centrality, after incremental node deletions, in a 10-regular graph, and Kademlia (k = 5, 15, 20) of 5000 (left side) and 15000 (right side) nodes. ................................................................. 39
3.5 Partitioning of graphs after removing nodes. The network becomes partitioned after removing on average about 40% of the nodes, in 10-regular graphs of size n=1000, to n=15000. ................................. 41
3.6 SOAPing attack: node T is under attack by the compromised node C and its clones. At each step one of the clones initiates the peering process with the T, until it is contained. After several iterations, the network is partitioned, and the botnet is neutralized.

3.7 SuperOnion construction, with $n = 5$, $m = 3$, $i = 2$.

4.1 Recent unexplained surge in the number of Hidden Services. The number of hidden services (.onion) suddenly tripled, before settling at twice the number before the surge.

4.2 Flow diagram of the honion system. We generate a set of honions to cover all the HSDirs and run a server behind each one. Here, we only show one descriptor per honion. When a visit happens to one of the honions, we can infer which HSDirs hosted it (and knew about its existence) using the consensus document and the list of relays. After identifying the potential suspicious HSDirs, we add the candidates to the bipartite graph.

4.3 ILP detection accuracy using simulation and the ground truth. Detection accuracy is beyond 80% as longs as the number of snooping HSDirs is less than 10%. The larger the set of snooping HSDirs, it is harder to detect the right snooper. The detection accuracy is above 95% as long as less than 4% of the HSDirs are malicious.

4.4 Plot of the visits to the honions. The daily onions show snooping HSDirs, before the “mystery” spike in hidden addresses. The number and intensity of the visits is increased after the spikes.

4.5 A typical graph representation of the visited honions, and the hosting HSDirs. The black nodes are the candidate malicious HSDirs calculated by the ILP algorithm. The gray nodes are the visited honions, and the colored nodes are all the HSDirs that hosted the honions, prior to a visit.

4.6 The global map of detected misbehaving HSDirs and their most likely geographic origin (in the interest of space we have omitted the only HSDir in Australia). The black icons represent the HSDirs hosted on a cloud platform that are exit nodes as well. The Red icons represent the nodes hosted on cloud that are not exit nodes, the blue icons represent the exit nodes that are not hosted on the cloud, and the green icons are the relays that are neither exit nodes, nor hosted on the cloud.
4.7 The behavior of the HSDirs. Some of them turned malicious after a while, and some disappeared shortly after the spikes in the number of hidden services.

4.8 The new trend of honion visits. The snoopers are delaying their visits to avoid identification.

5.1 Expected counts at HSDirs as a function of longevity. As we can see, 95% of the expected values are within two standard deviation of the mean expected value.

5.2 Histograms of expected values for different lifespans, 10 days, 30 days, 50 days, and 90 days. The results are based on 10000 runs of simulation. The Y-axis, represents the frequency percentage for each expected count.

5.3 The geolocation map of our relays. The green marker is CCIS NEU, the red markers are Online SAS, and the blue markers are Vultr servers. We distributed our servers to different geolocations for security and privacy reasons.

5.4 Distribution of the estimated longevity and raw longevity counts from the HSDirs.

5.5 CDF of distribution of the estimated longevity and raw longevity counts from the HSDirs.

5.6 Scatter plot of the popularity of hidden services as a function of their longevity.

5.7 Graphs of count matches between two entities, where each controls x relays. The X-Axis is the number of relays each entity controls, and the Y-axis, is the probability that both entities have a matching final count, with absolute difference of 0, 1, 2, and 3.

5.8 Count match between two parties, controlling up to 1000 (\(\frac{1}{3}\) of all relays), for a hidden service with longevity of 90 days. As we can see, the probability of a match decreases as we increase the number of control HSDirs.

5.9 Histogram of count differences between every two sets of HSDir entities.

6.1 Number of Tor users based on the statistics provided by Tor Metrics.
Chapter 1

Introduction

In the past few years, Tor and hidden services have become increasingly popular among privacy-concerned users. Specially, after revelations about large scale surveillance and data collection efforts by governments and corporations, more people are taking steps to protect their privacy online. Additionally, some users rely on Tor to evade censorship by ISPs and authorities to have free access to information.

The privacy and security protections that Tor provides, have attracted a large variety of users. For example, many legitimate users such as journalists, activists, and ordinary people use Tor to practice their right of privacy. Many mainstream services such as DuckDuckGo search engine, the New York Times, and Facebook, already have a presence on Tor. At the same time, Tor’s protections have also attracted illegal and malicious actors. For instance, ransomwares are a trivial subversion of basic cryptography primitives and cryptocurrencies such as Bitcoin. Botnets or online marketplaces for illegal drugs and contraband are other examples of abuse and subversion of privacy infrastructures. Tor and other privacy infrastructures have had a significant impact on society, protecting users, and are of significant importance to guarantee privacy. When such infrastructure is misused for launching malicious activities, such right becomes at risk.

In this thesis, we investigate the abuse of Tor from three different perspectives. We first describe OnionBots, the next generation botnets that rely on Tor and have self-healing network formation. Then, we introduce a new methodology and framework to detect the snooping Hidden Service Directories (HSDirs) inside
the Tor network. Finally, we study the lifespan of hidden services in a privacy-preserving manner to gain more insight into the dynamics of hidden services, without compromising the Tor’s safety and privacy standards. In the following, we briefly introduce these three aspects.

1.1 OnionBots: Next Generation Botnets

Botnets are a set of Internet connected nodes, that are under control of a botmaster. The botmaster sends his commands through a command and control (C&C) channel to the bots. Botnets are mostly used for distributed denial of service (DDoS) attacks and, ultimately, extortion of the domain owners. The first large-scale botnet can be traced back to the 2000s. Today, botnets are still a big economic and security problem on the Internet. In the arms race between the advancement of the detection and mitigation techniques, botnets have also evolved. The latest example of botnets advancement, is their take over of Internet of things (IoT) and embedded devices.

As we can see from Figure 1.1, the Tor relays’ bandwidth capability is at around 200 Gb/s, yet only 60% of this capacity is utilized. As the number of relays that participate in Tor network increases, the bandwidth capability of the Tor network also increases. The underutilized capacity of the Tor network attracts abuse from malicious actors. Specially, considering the privacy and security protection capabilities of the Tor network, combined with its bandwidth capacity, we are content that the next generation of botnets, which we call OnionBots, will subvert and abuse Tor.

The first generation of botnets that abused Tor (Skynet) [1] emerged from Reddit in 2012. Skynet was a central botnet, that only used hidden services for its C&C. In 2015, a variant of the Zeus malware that used Tor, started spreading. However, these botnets use Tor only to hide their C&C. In this thesis, we envision a botnet that uses Tor and hidden services capabilities beyond simply hiding the C&C. Current detection and mitigation of botnets, heavily rely on IP-based network properties and DNS traffic. However, Tor is an overlay network that operates independently from most Internet protocols. Therefore, these approaches do not work in the Tor network, and are not effective against OnionBots. Besides using Tor and hidden services to protect communication channels, OnionBots can achieve a low diameter and a low degree and be robust to partitioning under node
deletions. Such bots live symbiotically within the privacy infrastructures to evade detection, measurement, scale estimation, observation, and in general all current IP-based mitigation techniques.

1.2 HOnions: Exposing Snooping HSDirs

Tor’s safety and privacy rely on the assumption that a large majority of the relays are well behaving. It is, however, difficult to assess to what extent this condition holds true. The fact that many attacks are passive, makes it even harder to assess the significance of this threat. Previous work [2] have studied the existence and behavior of the malicious Exit nodes that intercept unencrypted traffic and perform man-in-the-middle attacks on the victims. However, the security of hidden services has not been studied before.

HSDirs are an integral part of the hidden services and are the equivalent of the DNS servers in the dark web. Each day the descriptors of hidden services, which contain information on how to contact them, are uploaded to a set of HSDirs. It is clearly indicated by the Tor Project that the HSDirs should not
snoop into the hidden services that are uploaded to them, since it is a breach of the user agreement and Tor’s privacy standards. In 2016, the still unexplained sudden surge in the number of hidden services (Figure 1.2), which tripled before returning to relatively smaller numbers, indicates that the Tor network is not well understood. This is in part due to Tor’s peer-to-peer nature, the privacy services it provides that limit measurements, and the attacks that it attracts.

We introduce Honey Onions (HOnions) as a framework to both estimate the number of snooping HSDirs and identify the most likely snooping HSDirs inside the Tor network. Our experimental results indicate that during the period of our work (72 days), at least 110 such nodes were snooping information about hidden services they host. We reveal that more than half of them were hosted on cloud infrastructure and delayed the use of the learned information to prevent easy traceback.

Figure 1.2: Number of unique hidden services.
1.3 Privacy Preserving Study of Hidden Services Lifespan

Data collection and study of networked systems is difficult, because of two aspects, technical and ethical. Privacy infrastructures such as Tor have even higher privacy risks, due to the nature of assumptions that are made, and inherent privacy dynamics of such network.

Currently, not much is known about the hidden services’ nature, and their lifespan. Studying the lifespan of hidden services can provide many benefits. For example, it allows investigation of the maliciousness of domains based on their lifespan. Short-lived hidden services are more likely not to be legitimate domains, as compared to long-lived domains. Previous work have looked at Tor and hidden services, without considering the required safeguards to protect the privacy of the users, and mitigate against its compromise. For example, in [3] the nature and dynamics of the hidden services are investigated by collecting onion addresses from a set of snooping HSDirs, and continuously probing them. Such work is clearly against Tor user agreement and ethics standards, and have been widely criticized by other researchers and the Tor Project.

In this work, we investigate the lifespan of hidden services by collecting data only from a small subset of the HSDir relays (2%) in a privacy-preserving manner. We develop simulation and extrapolation techniques to infer the lifespan of hidden services, using the collected data from a small number of relays, with high accuracy, while respecting the privacy and safety of the users and adhering to Tor’s ethics standards.

1.4 Outline

The rest of this thesis is organized as follows. In Chapter 2, we summarize the key mechanisms of Tor and hidden services’ architecture. We also provide a performance measurement analysis of hidden services. In Chapter 3, we look at the OnionBots, what we believe will be the next generation of resilient and stealthy botnets that abuse Tor. We introduce HOnions, a framework that exposes snooping HSDirs, in Chapter 4. We discuss our privacy preserving study
of hidden services lifespan in Chapter 5. Finally, in Chapter 6, we provide future research directions and outlook.
Chapter 2

Privacy Infrastructure: Tor

Tor [4] is an anonymity network that allows users to circumvent censorship and protect their privacy, activities, and locations from government agencies and corporations. This anonymity is achieved through a concept called onion routing, where the packets from source to destination are passed through a set of relays, and encrypted in layers using negotiated keys between the source and each relay. Figure 2.1 visualizes the onion routing concept with three layers. By default Tor uses three relays to establish a circuit. The first relay is called the Guard node, the second relay is the Middle node and the last relay is called the Exit node. To mitigate against traffic correlation attacks, no two relays from the same family group are used to create a circuit.

Besides anonymity for the clients, Tor also provides anonymity for the service providers with hidden services, which enables them to protect their location (IP address), yet allowing users to connect to them. Hidden services have been used to protect both legitimate services for privacy conscious users (e.g., Facebook), and for illicit purposes such as drug and contraband market [5], and extortion. This attracts attacks from a variety of actors. We first summarize some key mechanisms of Tor. In particular, we focus on the architecture of hidden services, both from the client and the service provider perspective.

2.1 Tor Hidden Services

The Tor hidden services architecture is composed of the following components:
Figure 2.1: Onion Routing, as it is used in Tor to establish circuits.

- **Server**, that runs a service (e.g., a web server).
- **Client**, that wishes to access the server.
- **Introduction Points (IntroP)**, a set of Tor relays, chosen randomly by the hidden service from the set of all relays. The IntroP forwards the initial messages between the server and the client’s Rendezvous Point. Note that the official Tor acronym for Introduction Points is IP. We use IntroP, as an acronym to avoid confusion with Internet Protocol (IP).
- **Rendezvous Point (RP)**, a Tor relay randomly chosen by the client, from the set of all relays. RP forwards the data between the client and the hidden service.
- **Hidden Service Directories (HSDir)**, a set of Tor relays chosen by the server to store its descriptors. The responsible HSDirs are chosen based on the fingerprint of their public keys, as described later.

**Server.** To enable access to a server, the service provider generates an RSA key pair. Then he calculates the SHA-1 digest of the generated public key, known as the **Identifier** of the hidden service. The .onion hostname is the base-32 encoding of the identifier. To connect to a hidden service, the aforementioned
Figure 2.2: Tor hidden service architecture and connection setup.

The identifier needs to be communicated to the clients through an external out-of-band channel. As depicted in Figure 2.2, the hidden service chooses a set of relays, called Introduction Points (IntroP), and establishes Tor circuits with them (step 1). After setting up the circuits, the hidden service calculates two service descriptors to determine which relays are the responsible HSDirs, using the following formula and uploads the descriptors to them (step 2).

\[
\text{descriptor-id} = H(\text{Identifier}||\text{secret-id-part})
\]
\[
\text{secret-id-part} = H(\text{time-period}||\text{descriptor-cookie}||\text{replica})
\]
\[
\text{time-period} = \frac{(\text{current-time} + \text{permanent-id-byte} \times 86400/256)/86400}{256}
\]

In the above equations, \(H\) is the SHA-1 hash digest. \text{Identifier} is the 80 bit truncated SHA-1 digest of the public key of the hidden service. \text{Descriptor-cookie} is an optional 128 bit field which could be used for authorization. The hidden services periodically change their HSDir. The \text{time-period} determines when each descriptor expires and the hidden services need to calculate the new descriptors and upload them to the new corresponding HSDirs. To prevent the descriptors from changing all at the same time, the \text{permanent-id-byte} is also included in the calculations. The \text{permanent-id-byte} is the first byte of the hash of the public key of a hidden service. The \text{Replica} index, takes values of 0 or 1, and results in two descriptors. Each descriptor is uploaded to 3 consecutive HSDirs, a total of 6. Note that the circle of HSDirs is sorted based on their fingerprint (SHA-1 hash of their public key) as shown in Figure 2.3. If the descriptor of a
hidden service falls between the fingerprint of HSDir\(_{k-1}\) and HSDir\(_k\), then it will be stored on HSDir\(_k\), HSDir\(_{k+1}\) and HSDir\(_{k+2}\).

The hidden service descriptor that is uploaded to an HSDir includes the information needed to contact the server, such as the IP address and port number of the Introduction Points. The descriptor also includes the signature of the hidden services, to ensure the authenticity of the information.

**Client.** When a client wishes to contact a hidden service, he first needs to compute the descriptor-id using the above formula, and contact the corresponding HSDirs (step 3). To communicate with a hidden service, the client first needs to choose a set of random relays as his Rendezvous Point (RP), and establish a circuit with them (step 4). Then he contacts the hidden service’s IntroPs to indicate his desire to contact the hidden service, and announcing his RPs (step 5). Then, the IntroP will forward this information to the hidden services (step 6). Finally, the hidden service establishes a circuit to the RPs, and the two can start communicating.

Note that in single hop hidden services, where the identity of the service provider is known and does not need to be protected (e.g., Facebook, The New York Times), only 3 hop circuits are established at the client side and the server is directly connected to the rendezvous point. These circuit formations increase the
network bandwidth and decrease the latency for connecting to hidden services. However, they are not widely used, since the majority of hidden services are established for the end point hiding and privacy.

### 2.2 Tor and Hidden Services Performance

To better understand the capability and scalability of Tor and hidden services’ architecture, we designed a set of experiments to measure their performance, efficacy, and scalability using different scenarios. First, we designed experiments to measure the performance of Tor interacting with an HTTP website as a baseline. Then we further investigated the performance of the hidden services (dark web). We study the performance trade-offs between the number of clients per Tor Onion Routers (OR), both in normal web and hidden services (dark web).

**Scenario 1:** In the first scenario, we consider using one client per Tor OR. We run $N$ wget programs over $N$ Tor ORs (i.e., 1 wget per 1 Tor OR) that carry traffic over Tor and interact with an external HTTP website. Torsocks can be used to torify any application that uses TCP streams. The architecture setup for scenario 1 is depicted in Figure 2.4a.

**Scenario 2:** In this case, we investigate running multiple ($N$) wget clients over 1 Tor OR. This experiment allows us to measure the optimal number of clients per Tor OR. Given the distributed nature of Tor and circuit establishment, each circuit is essentially limited by the bandwidth of its least capable relay in the circuit (this holds true in case of traffic to normal web, 3 hop hidden services, or 6 hop hidden services). This architecture setup is depicted in Figure 2.4b.

**Scenario 3:** After measuring the traffic from Tor to normal web, we move to measuring the performance of connections between Tor ORs on the client side and hidden services. In this scenario, each wget client uses 1 Tor OR on the client side. On the server side, only one onion service is running and it is hosted on one Tor OR. This would give us a baseline performance for hidden services. Note that in a connection to a hidden service, 6 hops are involved. Therefore, the importance of the circuit establishment and creation is slightly more impactful. Figure 2.4c shows our setup.

**Scenario 4:** After establishing a baseline performance measurement for hidden services, we investigate the degradation of bandwidth by increasing the number
Figure 2.4: Tor and hidden services’ performance measurement setup. We use different scenarios to measure the performance of Tor OR connecting to a normal HTTP server and also hidden services.
Figure 2.5: Tor and hidden services performance. Increasing the number of client connections on the same Tor OR, decreases performance. However, for the hidden services, each Tor OR can optimally support multiple hidden services.

Scenario 5: In this scenario we run $N$ hidden services, over $N$ Tor ORs (1 hidden service per 1 Tor OR). In this setup we can study the impact of distributing hidden services over several Tor ORs and also decrease the bias of circuit establishments per hidden service. Please refer to Figure 2.4e for this setup.

We run the simulations on two different Virtual Machines (VM), on a single server at Northeastern University. One VM runs the Tor client, and the other VM runs the servers. We run each simulation 10 times, and with increasing values of $N$. Figure 2.5, charts the results of our performance measurements. The X axis represents the number of Tor ORs ($N$), and the Y-axis represents the performance in KB/s. The bandwidth is per Tor relay. As we can see, the performance degrades sharply as we increase the number of clients on a single Tor OR connecting to the normal web. As we increase the number of Tor ORs and spread the clients equally over them, we can see that the performance does
not degrade. This can be explained because of the capacity of each circuit. As more clients connect to the normal web over single Tor OR, more circuits are not necessarily created. Note that in the case of hidden services, since there are already 6 hops in each circuit, the performance is on average less than normal web. However, as we can see by increasing the number of clients over a single Tor OR, the performance does not degrade. Distributing the increasing number of client over multiple Tor ORs has the same performance. Given that for each hidden service a new set of relays and circuits are established, each Tor OR is capable of hosting multiple hidden services, without sacrificing performance.

Please note in these experiment we used the default Tor settings both for the client and hidden services. Each client circuit has 3 randomly chosen relays. The circuits to the hidden services also have 3 randomly chosen relays (default values). It is possible to use one hop hidden service as a trade-off between speed and privacy. The detailed information on the Tor’s congestion control mechanism, and throttling are provided in [6]. We investigate the traffic that is completely over Tor. Some clients might use proxy servers to connect to hidden services, which adds more intermediary nodes to the connections and impacts the performance significantly.

2.3 Related Work

Previous work have studied Tor relay selection and circuit establishment and its impact on privacy and performance. For example, Tang and Goldberg [7] provide algorithms for Tor circuit scheduling that improve the latency of the circuits. Wacek et al. [8] explore the anonymity and performance trade-offs of the relay selection techniques in Tor. In other works [9, 10], authors investigate the effect of circuit selection on the privacy and deanonymization of Tor hidden services. Another body of work [11, 12] study the impact of dynamic circuit switching to reduce congestion in Tor. Since a large fraction of network capacity is consumed by the bulk of users’ traffic, the light interactive users’ experience degrades. The authors investigate the distribution of heavy and light traffic to identify the contributing factors to bottlenecks in circuits, to decrease end-to-end latency and increase the quality of communication in Tor.
2.4 Next Generation Hidden Services

The next generation hidden services [13], improve the security of hidden services and address many of the shortcomings of the current design. For example, the new design uses improved cryptography primitives (replaced SHA1/DH/RSA1024 with SHA3/ed25519/curve25519). The new design has an improved hidden service directory protocol that leaks less information (related to HOnions and longevity study, Chapter 4 and Chapter 5). The new protocol also makes targeted denial of service by HSDir placement infeasible (related to Chapter 3). Furthermore, it has better protection against impersonation and man-in-the-middle attacks [14] (our work that is not part of this thesis).

In the following we describe the details of the next generation hidden services, and how they mitigate against aforementioned shortcomings.

2.4.1 Shared Random Value

To mitigate against predetermined placement of a hidden service descriptors on HSDirs, the new design uses a commit-and-reveal protocol to generate fresh random number in the consensus document every 24 hours (at 00:00UTC). This random number is used to calculate the responsible HSDirs for the descriptors. Since the random number is not known to the public in advance, an adversary does not know the responsible HSDirs for any specific hidden service beforehand.

Before describing the protocol, we first need to define one important element of the Tor network: the authority directories (Authorities). These are 10 relays run by the core Tor people, and their IDs are hardcoded in the Tor software. These relays are responsible for generation of the consensus document. The consensus document contains information about all active relays in the Tor network.

The commit-and-reveal has two phases: 1) commit phase and 2) reveal phase. The commit phase happens every day during the first 12 rounds (00:00UTC to 12:00UTC). In this phase each authority server commits to a random value, and publishes it in the consensus document. In the reveal phase, which is the next 12 rounds (12:00UTC to 00:00 UTC), the authorities reveal the random numbers that they committed to previously. At the end of the 24 cycles, all authorities
calculate the shared random value (SRV) and publish it in the consensus document. The details of the process are as follows. \( RN \) is the SHA3 hashed value of a 256-bit random number. \( H \) is the SHA3 hashing function.

\[
\begin{align*}
\text{COMMIT} &= \text{base64-encode} (\text{TIMESTAMP} \mid H(\text{REVEAL})) \\
\text{REVEAL} &= \text{base64-encode} (\text{TIMESTAMP} \mid H(\text{RN}))
\end{align*}
\]

Based on the authority directories random values \( (RN) \), the random shared value is calculated as following.

\[
\begin{align*}
\text{HASHED\_REVEALS} &= H(ID_a \mid R_a \mid ID_b \mid R_b \mid \ldots) \\
\text{SRV} &= \text{SHA3-256}(\text{REVEAL\_NUM} \mid \text{VERSION} \mid \text{HASHED\_REVEALS} \mid \text{PREVIOUS\_SRV})
\end{align*}
\]

\( ID_a \) is the fingerprint of authority “a”, and \( R_a \) is the corresponding reveal value. \( \text{VERSION} \) is the protocol version number (currently version 1), and \( \text{PREVIOUS\_SRV} \) is the SRV value from the previous day, and if not known is set to NUL.

### 2.4.2 Encryption and Signing Keys

In the next generation of hidden services, a set of keys are derived from the master key (the master public key is the onion address) to perform various tasks. Two sets of these keys are of importance to us: the blinded signing key (used to sign the descriptor signing key) and the descriptor encryption key (a symmetric key used to encrypt the hidden service descriptors).

#### Blinded Signing Key

The blinded signing key is used to sign the descriptors signing key, and is derived from the identity key (onion address public key). The blinded signing
key changes periodically for additional safety and security. The following key blinding scheme is based on Ed255219.

The blinded signing key is derived for a nonce \( N \) and an optional secret \( s \), using the following approach.

\[
h = H(\text{BLIND\_STRING} | A | s | B | N)
\]

\[
\text{BLIND\_STRING} = "Derive temporary signing key"
\]

\[
N = "key-blind" | \text{period\_number} | \text{period\_length}
\]

\( B \) is the ed25519 basepoint [15]. The master key pair is \((a, A)\), where \( a \) is the private key and \( A \) is the corresponding public key \((A = a \cdot B)\). \text{period\_number} is calculated using the current consensus “valid-after”. \text{period\_length} is the length of a time period in minutes. The default time period length is 1440 (one day). The signatures are generated using EdDSA algorithm with the following “period keys”.

\[
a_t = h \cdot a \\
A_t = h \cdot A = (h \cdot a) \cdot B
\]

Descriptor Encryption Key

The hidden service descriptors are encrypted using a symmetric key derived from the onion address public key. If an HSDir does not have prior knowledge of the onion address (public key), he cannot decrypt the uploaded descriptors. This improvement protects against snooping HSDirs (Chapter 4). Furthermore, the longevity of hidden services cannot be investigated (Chapter 5), since the encryption of the descriptors changes every day. Therefore, an HSDir cannot check if the same hidden services is being uploaded.

The following is the detailed approach for the symmetric key derivation. \( H \) is the SHA-3 hash function. \text{public\_identity\_key} is the Ed25519 public key of
the hidden service. The **blinded-public-key** is the blinded period public key, as described previously.

\[
\text{subcredential} = H(\text{"subcredential" | credential | blinded-public-key})
\]
\[
\text{credential} = H(\text{"credential" | public-identity-key})
\]
Chapter 3

OnionBots: Next Generation Botnets

Over the last decade botnets rose to be a serious security threat. They are routinely used for denial of service attacks, spam, click frauds, and other malicious activities [16]. Both the research and industry communities invested a significant effort, analyzing, and developing countermeasures and products to effectively detect, cripple, and neutralize botnets. While some countermeasures operate on user computers, most are deployed at the ISP and enterprise levels. Many botnets were successfully neutralized by shutting down or hijacking their Command and Control (C&C) servers, communications channels (e.g., IRC), reverse engineering the algorithm used for the domain name generation (DGA) and preemptively blocking the access to these domains [17]. Such mitigation techniques exploit the fact that most current botnets rely on primitive communication architectures and C&C mechanisms. This forced botnet developers to continuously adapt; raising the level of sophistication of their design from the early static and centralized IRC or fixed servers’ Internet Protocol (IP) addresses to more sophisticated fast-fluxing [18] and even straightforward use of Tor hidden services [19, 20].

In this thesis, we are interested in investigating the next level of this arm-race. We contend that the next wave of botnets’ sophistication will rely on subverting privacy infrastructure and a non-trivial use of cryptographic mechanisms. The Tor project was very successful in building an infrastructure that protects users identity over the Internet and allowing one to host Internet servers without revealing her or his location using the Tor hidden services feature. Evidence of our predictions can be found in the malicious use of hidden services for hosting
the infamous silk road [5], instances of the Zeus [19] botnet, and the hosting of the CryptoLocker ransomware C&C server [21]. Interestingly, CryptoLocker combines Tor with the use of another privacy “infrastructure”, bitcoin the crypto currency, for the ransom payment. The combination of Tor and bitcoin makes it possible today to blackmail Internet users, anonymously be paid, and get away with it.

The current use of Tor and crypto-mechanisms in botnets is still in its infancy stages. Only hosting the C&C server as a hidden service still allows the detection, identification, and crippling of the communication between the bots. Recent research demonstrated that it is possible to successfully deny access to a single or few .onion server [22]. To assess the threat of crypto-based botnets, we advocate a preemptive analysis, understanding of their potential and limitations, and the development of mitigation techniques.

In this chapter, we present the design of a first generation of non-trivial Onion-Bots. In this OnionBot, communication is exclusively carried out through hidden services. No bot (not even the C&C) knows the IP address of any of the other bots. At any instant, a given bot is only aware of the temporary .onion address of a very small (constant) number of bots. Bots relay packets, but cannot distinguish the traffic based on their source, destination, or nature. At the same time, the bot master is able to access and control any bot, anytime, without revealing his identity. We show that this design is resilient to current mitigations and analysis techniques from botnet mapping, hijacking, to even assessing the size of the botnet. We also show that the proposed Neighbors-of-Neighbor graph maintenance algorithm achieves a low diameter, low degree, and high resiliency and repair in the event of a take-down (e.g., Tor DoSing or node capture/cleanup) of a significant fraction of the botnet nodes. Since our goal is to preemptively prevent the emergence of OnionBots, we also propose a novel mitigation technique against the Basic OnionBots. This technique exploits the same stealthy features of the OnionBot (namely peers not knowing each other’s identities) to neutralize the bots. The technique called Sybil Onion Attack Protocol (SOAP), gradually surrounds the bots by clones (or sybils) until the whole botnet is fully contained. Our goal is to draw the attention of the community to the potential of OnionBots and develop preemptive measures to contain them and ideally prevent their occurrence.

This chapter’s contributions are summarized as follows:
• A novel reference design for an OnionBotnet whose command, communication, and management are fully anonymized within the Tor privacy infrastructure.

• A communication topology with repair mechanisms that minimizes the nodes’ degree and maximizes resiliency.

• A performance evaluation and discussion of resiliency to various takedown attacks such as simultaneous denial of service attacks against multiple .onion botnet nodes.

• A Sybil mitigation technique, SOAP, that neutralizes each bot by surrounding it by clones.

• An outline and discussion of a set of techniques that can enable subsequent waves of Super OnionBots.

We first survey the current state of botnet design and mitigation techniques in Section 3.1. In Section 3.2, we present our proposed reference design for an OnionBotnet. We evaluate the resiliency and performance of the OnionBotnet, using several metrics in Section 3.3. We finally investigate potential mechanisms to prevent the rise of such botnets in Section 3.4. In Section 3.5, we outline a set of techniques that can enable subsequent waves of Super OnionBots.

3.1 Related Work

We first review the evolution of botnets and why we believe the next generation of botnets would subvert privacy infrastructures to evade detection and mitigation. Currently, bots are monitored and controlled by a botmaster, who issues commands. The transmission of these commands, which are known as C&C messages, can be centralized, peer-to-peer or hybrid [23]. In the centralized architecture the bots contact the C&C servers to receive instructions from the botmaster. In this construction the message propagation speed and convergence is faster, compared to the other architectures. Furthermore, it is easy to implement, maintain and monitor. However, it is limited by a single point of failure. Such botnets can be disrupted by taking down or blocking access to the C&C server. Many centralized botnets use IRC or HTTP as their communication channel. GT-Bots, Agobot/Phatbot [24], and clickbot.a [25] are examples of
such botnets. A significant amount of research focused on detecting and blocking them [26], [27, 28, 29, 30, 31]. To evade detection and mitigation, attackers developed more sophisticated techniques to dynamically change the C&C servers, such as: Domain Generation Algorithm (DGA) and fast-fluxing (single flux, double flux).

Single-fluxing is a special case of fast-flux method. It maps multiple (hundreds or even thousands) IP addresses to a domain name. These IP addresses are registered and de-registered at rapid speed, therefore the name fast-flux. These IPs are mapped to particular domain names (e.g., DNS A records) with very short Time To Live (TTL) values in a round robin fashion [18]. Double-fluxing is an evolution of single-flux technique, it fluxes both IP addresses of the associated fully qualified domain names (FQDN) and the IP addresses of the responsible DNS servers (NS records). These DNS servers are then used to translate the FQDNs to their corresponding IP addresses. This technique provides an additional level of protection and redundancy [18]. Domain Generation Algorithms (DGA), are the algorithms used to generate a list of domains for botnets to contact their C&C. The large number of possible domain names makes it difficult for law enforcements to shut them down. Torpig [32] and Conficker [33] are famous examples of such botnets. Figure 3.1, from The Honeynet Project [34] depicts the single-flux and double-flux network setup and architecture.

A significant amount of research focuses on the detection of malicious activities from the network perspective, since the traffic is not anonymized. For example [35, 36, 37, 38, 39, 40] inspect the DNS traffic and use machine learning clustering and classification algorithms to detect benign and malicious traffic and cluster the malicious traffic. BotFinder [41] uses the high-level properties of the bot’s network traffic and employs machine learning to identify the key features of C&C communications. DISCLOSURE [42] uses features from NetFlow data (e.g., flow sizes, client access patterns, and temporal behavior) to distinguish C&C channels. Other work [43, 44] focus on endpoints’ static metadata properties and the order of the high-level system events for threat classification.

The next step in the arms race between attackers and defenders was moving from a centralized scheme to a peer-to-peer C&C. Storm [45], Nugache [46], Walowdac [47] and Gameover Zeus [48] are examples of such botnets. Some of these botnets use an already existing peer-to-peer protocol, while others use customized protocols. For example, earlier versions of Storm used Overnet, and the
new versions use a customized version of Overnet, called Stormnet [45]. Meanwhile, other botnets such as Walowdac and Gameover Zeus organize their communication channels in different layers.

Previous works studied specific mitigations against peer-to-peer botnets. Many of the works use properties of benign peer-to-peer network formation and dynamics to distinguish the malicious and benign traffic and network formation. For example, BotGrep [49] uses the unique communication patterns in a botnet to localize its members by employing structured graph analysis. Zhang et al. [50] propose a technique to detect botnet P2P communication by fingerprinting the malicious and benign traffic. Yen and Reiter [51] use three features (peer churn, traffic volume and differences between human-driven and bot-driven behavior) in network flow to detect malicious activity. Coskun et al. [52] propose a method to detect the local members of an unstructured botnet by using the mutual contacts. As we can see, some of these techniques rely on observing the unencrypted traffic. Therefore, by using a privacy infrastructure such as Tor they can be evaded. Furthermore, these techniques assume that the structure and network formation can be observed. Tor decouples the IP address from the host, and a host can use many hidden services as communication interface with no additional cost.
Another category of botnets subvert platforms such as URL shortening services, and online social networks (e.g., Twitter) for their C&C. Some of these alternative botnet designs are proposed by researchers, and not deployed over the Internet yet.

For example, Kartaltepe et al. [53], investigate a new generation of botnets that use online social networks, such as Twitter for their C&C infrastructure. An instance of such malware, Naz, gets its commands by making GET requests to the RSS feed of its botmaster on Twitter. The tweets contain the base64 encoding of shortened URLs (e.g., bit.ly) that redirect the bot to the compressed malicious payload. Lee and Kim [54], explore the design and mitigation of botnets that use URL shortening services (USSes) for alias fluxing. A botmaster uses the USSes to hide and obfuscate IP address of the C&C by using a dictionary of 256 words for each part of an IPv4. For example, 10.15.43.89 can be mapped to “Brown.Fox.Jumps.Over.” Then this expression is transformed into a search query, such as google.com/q?=Brown+Fox+Jumps+Over. Using the URL shortening service, bots can find the corresponding IP address by using the same dictionary. Nappa et al. [55], propose a parasitic botnet protocol that exploits Skype’s overlay network. Skype provides a widespread resilient network with a large install base for C&C infrastructure. The communications between the master and the bots are encrypted using adhoc schemes. The protocol broadcasts messages to all peers in the network, similar to the algorithms used in Gnutella. Once each peer receives a new message, it is passed to all of the peer’s neighbors. Xu et al. [56] study the use of DNS for C&C using two communication modes to piggyback messages over the DNS messages, codeword and tunneled. In the codeword mode, the bot makes a query (e.g., codeword.example.com) and the server replies with an appropriate answer (e.g., the IP address of a victim for DoS attack). In the tunneled mode the client encodes its data using a base32 encoding and sends a CNAME query. After receiving the query, the server uses base32 encoding to construct the corresponding CNAME reply.

Very recently, the use of Tor received more attention from malware and botnet authors. For example, the new 64-bit Zeus employs Tor anonymity network in its botnet infrastructure [19]. It creates a Tor hidden service on the infected host and the C&C can reach these infected hosts using their unique .onion address through Tor. Another example is ChewBacca [20], which uses Tor, and logs the keystrokes of the infected host and reports them back to the botmaster. The C&C is an HTTP server that is hosted as a hidden service. Although using Tor and hidden services makes the detection and mitigation more difficult, these bots
are still using the basic client-server model. This leaves them open to single point of failure.

3.2 OnionBot: a Cryptographic P2P Botnet

In this section, we look at the details of the proposed OnionBot, a novel non-IP based construction that is immune to the current mitigation techniques. We explore different bootstrapping approaches and a distributed, self-healing, low-degree, low-diameter overlay peer-to-peer network formation.

3.2.1 Overview

OnionBot retains the life cycle of a typical peer-to-peer bot [57]. However, each stage has unique characteristics that make OnionBots different from current peer-to-peer botnets. As a result, existing solutions are not applicable to OnionBots. For example, in the infection stage, each bot creates a .onion address and generates a key to encrypt the messages. In the rally stage, the bots dynamically peer with other bots that are the foundation of a self-healing network. Furthermore, at the waiting stage, bots periodically change their address to avoid detection and mitigation. These new .onion addresses are generated from the key that is shared with the botmaster. This allows the botmaster to access and control any bot through the shared key, anytime, without revealing his identity.

**Infection**: this is the first step in recruiting new bots. It can happen through traditional attack vectors such as phishing, spam, remote exploits, drive-by-download or zero-day vulnerabilities [58]. A great body of literature have focused on different spread mechanisms [59, 60, 61]. In this work we focus on the remaining stages of a bot’s life cycle.

**Rally**: in order to join a botnet, the newly infected bots need to find the already existing members of the network. In peer-to-peer networks this process is called bootstrapping. For clarity reasons we use the same terminology in describing OnionBots. Based on the requirements of the network, the complexity and flexibility of bootstrapping techniques vary significantly. OnionBots necessitate a distributed mechanism to maintain a low-degree network. Such requirements, demands a bootstrapping mechanism that is able to evolve with the
network. OnionBots benefit from the decoupling of IP address and bots, which allows them to address such requirements. In section 3.2.2 we discuss different techniques and their ramifications in more detail.

**Waiting**: in this state, a bot is waiting for commands from the botmaster. Generally the command transmissions can be pull-based (bots make periodic queries to the C&C) or push-based (botmaster sends the commands to the bots), and there are trade-offs in each mechanism. For example, in the pull-based approach, if bots aggressively make queries for the C&C messages, it allows faster propagation of commands. However, it results in easier detection of C&C and the bots. In the push-based approach, it is important to be able to reach each bot within a reasonable number of steps. Furthermore, to prevent leakage of information about the botnet operation and topology, it should not be feasible for an adversary to distinguish the source, destination and the nature of the messages. Meanwhile, satisfying such requirements is not trivial in self-healing networks. Later in section 3.2.4 we discuss how in OnionBots, the botmaster is able to access and control any bot, at any time.

**Execution**: at this stage the bots execute the commands given by the botmaster (e.g., bitcoin mining, sending spam [62] or DDoS attack [63], [64]), after authenticating them. Recently, botmasters started offering botnet-as-a-service as it was previously predicted by researchers in 2008 [65]. Considering that the OnionBots make use of cryptographic primitives beyond the basic, trivial encryption/decryption of payloads, it allows them to offer the botnet for rent. In section 3.2.5, we explain how this can be done, in a distributed way, and without further intervention of the botmaster.

In the next sections we will focus on describing the key mechanisms of OnionBots.

### 3.2.2 Bootstrap

As mentioned previously the bootstrapping is an essential part of network formation in peer-to-peer networks. Additionally, in OnionBots it provides the foundation for the self-healing graph construction. In the following, we study different approaches and their trade-offs. We discuss how these concepts should be adapted to the context of a privacy infrastructure such as Tor. Note that the
address of a peer list in our protocol refers to the .onion address of the peers, unless stated otherwise.

- **Hardcoded peer list**: in this setting each bot has a list of other bots to contact at the beginning. Since the infections can be carried out by the bots, the new peer lists can be updated. Each peer upon infecting another host sends a subset of its peer list. Each node in the original peer list will be included in the subset with probability \( p \). In the conventional botnets this scheme is vulnerable to detection and blacklisting, given IP addresses cannot be changed easily. However, in OnionBots, the .onion address is decoupled form IP address, and changes periodically as it is described in section 3.2.4. Therefore, the current mitigation techniques are not applicable.

- **Hotlists (webcache)**: this is conceptually similar to the hardcoded peer list. However, each bot has a list of peers to query for the other peers. In this setting, the adversary (defenders) will only have access to a subset of servers, since each bot only has a subset of the addresses, and these subsets can be updated upon infection or later in the waiting stage.

- **Random probing**: in this scheme a bot randomly queries the list of all possible addresses, until it finds a bot listening on that address. Although such approach can be used in IPv4 and IPv6 [66] networks, it is not practical in the context of Tor .onion addresses. Since the address space is much larger than IPv4 and IPv6 (to craft an address with specific first 8 letters, it takes about 25 days [67]); Randomly querying all possible .onion addresses requires probing an address space of size \( 2^{80} \) (the length of onion domains is 80 bits).

- **Out-of-band communication**: the peers list can be transmitted through another infrastructure. For example, by using a peer-to-peer network such as BitTorrent and Mainline DHT, to store and retrieve peer lists, or by using social networks, such as Twitter, Facebook or YouTube. Other infrastructures can also be subverted as means of communication. To avoid detection, covert channels can be implemented on different infrastructures.

We envision that OnionBots would use a customized approach based on hardcoded peer list and hotlists. As mentioned earlier, in OnionBots the blacklisting of nodes is not practical, since their addresses change periodically, because of the IP decoupling. In the following section, we describe how OnionBots address the bootstrapping and recruitment during network formation and maintenance.
3.2.3 Maintenance of the OnionBot Communication Graph

OnionBots form a peer-to-peer, self-healing network that maintains a low degree and a low diameter with other bots to relay messages. Peer-to-peer networks are broadly categorized into structured and unstructured [68], where both categories are used by botnets, and are studied in previous works [46, 45, 69]. Structured network formation are proved useful for information retrieval and storage. At the same time, it makes them vulnerable to mapping, and ultimately take down. The already existing peer-to-peer networks are generic in terms of their operations. Therefore, their design and resiliency are based on different assumptions and requirements. In the following, we propose a Dynamic Distributed Self Repairing (DDSR) graph, a new peer-to-peer construction that is simple, stealthy and resilient. It is also an overlay network, formed over a privacy infrastructure such as Tor.

Neighbors of Neighbor Graph: In this section, we introduce DDSR graph construct that is used in the network formation of OnionBots. The proposed construct is inspired by the knowledge of Neighbors-of-Neighbor (NoN). Previous work [70] studied the NoN greedy routing in peer-to-peer networks, where it can diminish route-lengths, and is asymptotically optimal. In this work we discuss how NoN concepts can be used to create a self-healing network.
Consider graph $G$ with $n$ nodes ($V$), where each node $u_i \in V$, $0 \leq i < n$, is connected to a set of nodes. The neighbors of $u_i$, are denoted as $N(u_i)$. Furthermore, $u_i$ has the knowledge of nodes that are connected to $N(u_i)$, meaning that each node also knows the identity of its neighbor's neighbors. In the context of our work, the identity is the .onion address.

**Repairing:** When a node $u_i$ is deleted, each pair of its neighbors $u_j, u_k$ will form an edge $(u_j, u_k)$ if $(u_j, u_k) \notin E$, where $E$ is the set of existing edges (Algorithm 1). Figure 3.2 depicts the node removal and the self repairing process in a 3-regular graph with 12 nodes. The dashed red lines indicate the newly established links between the nodes. For example, as we can see if we remove one of the nodes (7), its neighbors (0, 1, 4) start to find a substitute for the deleted node (7), to maintain the aforementioned requirements. In this case the following edges are created: (0, 1), (1, 4), and (1, 4).

The basic DDSR graph outlined in the previous paragraph, does not deal with the growth in the connectivity degree of each node, denoted by $d(u)$. After multiple deletions, the degree of some nodes can increase significantly (further discussed in Section 3.3). Such increase of the nodes' degree is not desirable for the resiliency and the stealthy operation of the botnet. Therefore, we introduce a pruning mechanism to keep the nodes' degree in the range $[d_{\text{min}}, d_{\text{max}}]$. we set $d_{\text{min}} = 5$ and $d_{\text{max}} = 15$. Note that $d_{\text{min}}$ is only applicable as long as there are enough surviving nodes in the network. The pruning helps with the maintenance of the network formation. It also enables limiting the amount of information that each node holds and the information that is shared with other nodes.

**Pruning:** Consider the graph $G$, when a node $u_i$ is deleted, each one of its neighbors, starts the repairing process (Algorithm 1). However, this scheme causes the degree of the neighbors of node $u_i$, to increase significantly after $t$ steps (deletions). To maintain the degree in the aforementioned range $[d_{\text{min}}, d_{\text{max}}]$, each neighboring node of the deleted node $(u_i)$ deletes the highest degree node from its peer list. If there is more than one such candidate, it randomly selects one among those for deletion, until its degree is in the desired range. Removing the nodes with the highest degree, ensures that the graph remains connected. Furthermore, it mitigates against formation of critical hub nodes, and increases the resiliency of the resulting graph formation to a single point of failure and take down. This can be viewed as a variant of the degree-constrained spanning tree,
which is NP-hard [71]. It is the problem of finding a spanning tree, which is of minimal length, and the vertices’ degrees do not exceed a given maximum. We use a greedy heuristic approach, for its simplicity.

\begin{algorithm}
\begin{algorithmic}
\State \textbf{Input:} DN: List of deleted node’s neighbors
\State \hspace{1em} CN: List of current neighbors
\State \textbf{Output:} PN: List of new neighbors
\State \hspace{1em} \textbf{1} \hspace{1em} PN \leftarrow \emptyset
\State \hspace{1em} \textbf{2} \hspace{1em} \textbf{foreach} \hspace{1em} dn \in DN \hspace{1em} \textbf{do}
\State \hspace{2em} \textbf{3} \hspace{2em} PN[dn] \leftarrow \text{deg}(dn)
\State \hspace{1em} \textbf{4} \hspace{1em} \textbf{end}
\State \hspace{1em} \textbf{5} \hspace{1em} \textbf{foreach} \hspace{1em} cn \in CN \hspace{1em} \textbf{do}
\State \hspace{2em} \textbf{6} \hspace{2em} PN[cn] \leftarrow \text{deg}(cn)
\State \hspace{1em} \textbf{7} \hspace{1em} \textbf{end}
\State \hspace{1em} \textbf{8} \hspace{1em} \textbf{while} \hspace{1em} \text{len}(PN) > \text{deg}_{\text{max}} \hspace{1em} \textbf{do}
\State \hspace{2em} \textbf{9} \hspace{2em} \text{delete potential neighbor with highest degree}
\State \hspace{1em} \textbf{10} \hspace{1em} \textbf{end}
\end{algorithmic}
\caption{Repairing and pruning algorithm after node deletion}
\end{algorithm}

**Forgetting:** In the proposed OnionBot, nodes forget the .onion address of the pruned nodes. Additionally, to avoid discovery, mapping, and further blocking, each bot can periodically change his .onion address and announce the new address to his current peer list. The new .onion address is generated based on a secret key and time. This periodic change is possible because of the decoupling between IP address and the bots, which is provided by Tor. Later in section 3.2.4, we explain how the C&C is able to directly reach each bot, even after they change their addresses.

**Bot to Bot Communication:** The bots use the flooding broadcast approach to relay the messages that they receive from any node (Algorithm 2). The bots send the received messages to all their neighbors, after they verify they have not received the message previously. This can be achieved by storing a hash of received messages, or storing the counter index of the last received message.

### 3.2.4 Command and Control Communication

In this section, we show how Tor enables a stealthy and resilient communication channel with C&C. As mentioned before, OnionBot is a non IP address based
Input: $M$: message received from another node  
CN: List of current neighbors

1. if message $M$ is new and not forwarded already then
2. foreach $cn \in CN$ do  
3. forward $M$ to $cn$
4. end

Algorithm 2: Flooding communication between a bot and its peers.

In OnionBots we assume two classes of messages: 1) messages from C&C to the bots, and 2) messages from bots to C&C. The messages from C&C can be either directed to an individual node(s) (e.g., a maintenance message telling a bot to change its peers) or directed to all bots (e.g., DDoS attack on example.com). If a message is directed at an individual bot, $b$, the botmaster sends $\{H(K_b)\}_{PR_{CC}}$, and if the message is directed at all bots the botmaster sends $\{H(K_{ephemeral})\}_{PR_{CC}}$. $K_b$ is the key that is shared between an individual bot and the botmaster (further explained below). $H$ is a hash function, and $PR_{CC}$ is the private key of the botmaster.

The botmaster can also setup group keys to send encrypted messages to a group of bots. While a bot can tell the difference between a broadcast message and messages directed to an individual bot(s), it is not able to identify the source, the destination, and the nature of these messages. If a bot is an intended target audience of a message, he is able to decrypt the message; because the botmaster is either sending the $\{H(K_{ephemeral})\}_{PR_{CC}}$ or $\{H(K_b)\}_{PR_{CC}}$. If a botnet is the individual target or a member of the target group, he has knowledge of the $K_b$, and can check if the $H(K_b)$ matches the hash of his key. Therefore, the authorities are not able to detect different messages and drop harmful message and only allow the maintenance message to pass through. As a result, they cannot create the illusion that the botnet is operational, when it is actually neutralized.

In OnionBots, the bots report their addresses to C&C, and establish a unique key to be shared with the botmaster at the infection/rally stage. Each bot generates a symmetric key, $K_b$, and reports it to the C&C by flooding the network. $K_b$ is encrypted with the hardcoded public key of the C&C ($\{K_b\}_{PK_{CC}}$). This allows C&C to have direct access to the bots, even after they change their .onion address. After establishing the key, bots can periodically change
their .onion address based on a new private key generated using the following recipe: \texttt{generateKey}(PK_{CC}, H(K_b, i_p)). Where, \texttt{generateKey} is a deterministic prime number generator (Algorithm 3). \(H\) is a hash function, and \(i_p\) is the index of period (e.g., day). All messages are of the same fixed size, as they are in Tor.

\begin{algorithm}
\textbf{Input:} \(PK_{CC}\): public key of the C&C, \(K_b\): shared secret
\hspace{1cm} \(i_p\): period index, “p” or “q”
\textbf{Output:} Two prime numbers \((p, q)\) to be used to generate RSA key pair
1. \(c \leftarrow 0\)
2. \(h \leftarrow NUL\)
3. \textbf{while} \(c \leq i_p\) \textbf{do}
4. \hspace{1cm} \(h \leftarrow H(h, PK_{CC}, H(K_b, i_p), “p”)\)
5. \hspace{1cm} \(c = c + 1\)
6. \textbf{end}
7. \(p \leftarrow \text{first prime number} \geq h\)
8. \(c \leftarrow 0\)
9. \(h \leftarrow NUL\)
10. \textbf{while} \(c \leq i_p\) \textbf{do}
11. \hspace{1cm} \(h \leftarrow H(h, PK_{CC}, H(K_b, i_p), “q”)\)
12. \hspace{1cm} \(c = c + 1\)
13. \textbf{end}
14. \(q \leftarrow \text{first prime number} \geq h\)
\end{algorithm}

**Algorithm 3:** \texttt{generateKey} for generation of two prime numbers \(p\) and \(q\) to be used in the RSA key generation. \(H\) is a hash function to produce 512 bit output.

To summarize, the botmaster has the knowledge of hashing function, \(H\), \texttt{generateKey}, his private, public key pair, \((PR_{CC}, PK_{CC})\), and \(K_b\), where \(K_b\) is a shared key between a botnet/group botnets and the botmaster. A botnet/group of botnets, have knowledge of the hashing function, \(H\), \texttt{generateKey}, \(K_b\) and \(PK_{CC}\).

### 3.2.5 Operation

While many current botnets lack adequate secure communications [72] (e.g., sending messages in plaintext) that leaves them open to hijacking, the OnionBot’s communication is completely encrypted since it uses Tor and SSL. Note that the encryption keys are unique to each link. For example, Miner does not encrypt
Table 3.1: Cryptographic use in different botnets. Data from [72]

<table>
<thead>
<tr>
<th>Botnet</th>
<th>Crypto</th>
<th>Signing</th>
<th>Replay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Miner</td>
<td>none</td>
<td>none</td>
<td>yes</td>
</tr>
<tr>
<td>Storm</td>
<td>XOR</td>
<td>none</td>
<td>yes</td>
</tr>
<tr>
<td>ZeroAccess v1</td>
<td>RC4</td>
<td>RSA 512</td>
<td>yes</td>
</tr>
<tr>
<td>Zeus</td>
<td>chained XOR</td>
<td>RSA 2048</td>
<td>yes</td>
</tr>
</tbody>
</table>

its traffic, and botnets such as Storm, ZeroAccess and Zeus use simple algorithms that mostly obfuscate the traffic instead of fully encrypting the traffic. At the same time, they do not provide signatures to provide authentication of the messages, or use weak and broken key sizes. Furthermore, they do not provide mitigations against replay attacks. Table 3.1 summarizes a number of botnet families and their lack of adequate cryptographic blocks [72]. Furthermore, we introduce new cryptographic blocks that enable the OnionBot to offer new services, such as a botnet-for-rent [65] and a distributed computation platform for rent.

To achieve the aforementioned services, we need to account for three aspects of the messages: 1) authenticity, 2) expiration time, and 3) legitimacy. Public key encryption and certificates that are based on chain of trust, are suitable candidates to solve authenticity and legitimacy of the messages. Expiration of the message (rental contract term) can be addressed by using timestamps. In the following, we describe the details of such operation.

Imagine Trudy wishes to rent the botnet from Mallory, and every bot has the public key of Mallory hardcoded. Trudy sends her public key $PK_T$ to Mallory, to be signed by the private key of Mallory $SK_M$. The signed message ($T_T$) acts as a token containing $PK_T$, an expiration time, and a list of white-listed commands. When Trudy wants to issue a command to her rented bots, she signs her command by using her private key $SK_T$ and includes $T_T$. This way, the bots are able to verify the legitimacy of such commands, by looking at the token and the signature of the message.

As a business operation, Trudy pays Mallory using Bitcoin, where the whole transaction can be carried out over Silk Road 2.0. Furthermore, Mallory can instruct her bots to install computation platforms such as Java Virtual Machine (JVM). By doing so, she can also offer a cross-platform distributed computation infrastructure to carry out CPU intensive tasks, such as bitcoin mining or password cracking. The rise of cryptocurrencies, specially those that do not rely on
proof of work (PoW) hashing functions that can be accelerated by use of dedicated hardware, GPUs, and Application-specific integrated circuit chips (ASIC), can further motivate the botmaster to deploy his botnet for mining.

3.3 OnionBots Evaluation

To evaluate the envisioned OnionBots, we look at two aspects: the self-healing network formation resilience performance and the resilience to analysis techniques, such as botnet mapping, hijacking, or even assessing the size of the botnet. The NoN look-ahead routing is proven to be asymptotically optimal. However, such claims have not been studied in the context of self-healing networks. Moreover, we compared the DDSR construct with Kademlia, a simple and widely used peer-to-peer network formation. In this work we use empirical data and simulations for evaluation of OnionBots.

3.3.1 Mapping OnionBot

OnionBots provide a more resilient structure by using features available in the Tor network that previous botnets lack. In addition to many security and privacy services that Tor provides, it also enables a plethora of operational services. For example, all OnionBot nodes are directly accessible, even those running behind NAT, because of the NAT traversal capability that are implemented with the hidden services, compared to previous work [73]. Furthermore, if a bot is captured and the address (.onion) of other bots is revealed, it is still not practical to block the bots. Additionally, bots can periodically change their .onion addresses at no cost, because of the IP addresses decoupling, and share it only with their operational peers, to limit the exposure of the bot’s address. As a result, one single .onion could be blocked, as we later discuss in section 3.4, but it is not feasible to block all of them. Please note that by moving to the next generation of Tor hidden services, blocking a specific onion address is also not feasible anymore, since the responsible HSDirs are not known in advance, given the nondeterministic seed value that is updated and used daily to assign the responsible HSDirs for a specific hidden service. For a more detailed description of the next generation hidden services, and its impact on our work, please refer to Chapter 2.
Figure 3.3: The average closeness centrality (3.3a, 3.3b) and degree centrality (3.3c, 3.3d) of nodes in a $k$-regular graph, ($k = 5, 10, 15$) with 5000 nodes after 30% node deletions, with and without pruning.
3.3.2 OnionBot Network Resilience

To evaluate the resiliency and performance of the self repairing construction we use some of the metrics that are used in graph theory, such as the changes in graph centrality after node deletions. Centrality metrics examined in previous studies [74], [75] include closeness centrality and degree centrality. These metrics are also used to study social networks, since they provide indications of the importance of each node within the network formation.

The objective of OnionBot network formation is to avoid the formation of a central node, since they would become the single point of failure. Note that in a central botnet, the C&C is the single point of failure. In the OnionBot formation there is no central botmaster C&C. However, the peer-to-peer network needs to be resilient against partitioning attacks. Otherwise, the messages do not reach all nodes. To partition a network it is more efficient to perform targeted node deletion, instead of random node deletions. The most effective approaches are targeting the nodes with the highest degree, or the nodes with highest betweenness centrality [76]. By avoiding the formation of a central node, we decrease the efficiency of the node removal strategies. At the same time, the network formation needs to have desirable characteristics such as fast message propagation and connectivity.

Closeness Centrality: The closeness centrality of node $u$, is the inverse of sum of the length of the shortest paths between node $u$ to all $n - 1$ other nodes. Since the sum of distances depends on the number of nodes, it is normalized.

$$C(u) = \frac{n - 1}{\sum_{v \neq u} d(u, v)}$$

Where $n$ is the number of nodes, and $d(u, v)$ is the length of the shortest path between nodes $u$ and $v$. It is an indication of how fast messages can propagate in the network, from a node, $v$, to all other nodes sequentially. A central node is a node that is close, on average, to all other nodes. Therefore, nodes with highest closeness centrality can act as broadcasters to the entire network, since they can propagate a message faster than all other nodes. Since we are using the flooding mechanism to propagate message in the network, broadcasters are the nodes that can relay information to a large number of nodes.
**Degree Centrality**: The *degree centrality* of a node $u$ is the number of its neighbors. The degree centrality values are normalized by the maximum possible degree in graph $G$. Therefore it is the fraction of nodes that $u$ is connected to. It is an indication of immediate chance of receiving whatever is flowing through the network (e.g., messages). The degree centrality also indicates the most connected node, or the hub. For a stealthy botnet that can be used for advanced persistent attack (APT), a low degree centrality is required, to avoid detection. Detection of botnets, results in take over and mitigation measures by law enforcements and ISPs.

In the following, we resort to simulations to get a good sense of the properties of OnionBot. We first compare the impact of pruning on the efficacy of OnionBots, by simulating the random node deletion process in a $k$-*regular* graph, $(k = 5, 10, 15)$ of 5000 nodes, with up to 30% (1500) node deletions. As discussed, a targeted node deletion is more efficient to partition network formations. However, it needs a complete view of the network, which adversaries do not have in OnionBots. Furthermore, since the OnionBots network structure avoids the formation of central and hub nodes, the targeted node deletion are not applicable. Therefore, we use the random node deletion, which is a more realistic model of the real world operations against botnets. Additionally, we follow by comparing the OnionBot with Kademlia using simulations. Kademlia is a popular structured peer-to-peer network that is also simple to implement. The results are based on average of 5 runs of simulation per scenario.

Figure 3.3 illustrates the average closeness centrality with pruning (Figure 3.3b) and without pruning (Figure 3.3a). As we can see in Figure 3.3a, closeness centrality of the nodes is stable, and it does not increase during the node deletion. This is due to the fact that pruning mechanism enforces the DDSR to maintain a bounded range of neighbors. Furthermore, we measure the degree centrality of the nodes in the aforementioned graph, with pruning (Figure 3.3d) and without pruning (Figure 3.3c). As we can see, without pruning the degree of nodes increases significantly during the node deletions. This is because of the introduction of additional neighbors in the repairing stage. Low degree centrality is desirable in advanced persistent attacks, since it decreases the chances of detection and take down, by maintaining a low profile.

To better understand the effects of size on the performance and resiliency of OnionBots, we simulate a small botnet of size 5000 [74] and a medium botnet of size 15000. Furthermore, we also simulate the Kademlia network formation with
\( k = 5, 15, 20 \) for comparison. Figure 3.4 depicts degree centrality and number of connected components after node deletion for the Kademlia and OnionBot DDSR network formations. In the following, we provide a brief description of Kademlia network formation.

**Kademlia Peer-to-Peer Network**

Kademlia is a distributed hash table (DHT) for structured peer-to-peer overlay network formation. It is widely used in a variety of peer-to-peer networks, because of its speed and ease of implementation. For example, BitTorrent, Ethereum, Kad Network, Gnutella DHT, and InterPlanetary File System (IPFS) use Kademlia DHT formation.

In Kademlia, each node has a unique ID and stores a routing table of its neighbors. In the following, we briefly describe the routing tables, lookup and joining processes. We first need to define a “distance” metric in the network. In case of Kademlia, the XOR function is defined as the distance between two nodes with ID_1 and ID_2.

**Routing Tables:** Each node has \( n \) lists to store its neighbors, where \( n \) is the number of bits in the nodes’ ID. For example nodes with 128 bit IDs, have 128 lists, one for each bit. These lists store the information about other nodes that have distance \( i \) (\( \forall i \in \{0 \ldots n\} \)) with the node (i.e., having the first different value in their IDs at the \( i^{th} \) bit). Each list is referred to as a \( k \)-bucket, where \( k \) is a system wide parameter. Each \( k \)-bucket is a list having up to \( k \) entries inside. As we can see the number of nodes that are stored in each \( k \)-bucket decreases quickly.

**Lookup:** To lookup a node or resource, each participant looks in its own \( k \)-buckets that are closest to the target. If the target is not already in its lists, the node sends a query to its neighbor in the closest \( k \)-bucket. This process repeats until it finds the target, or the iterations are over.

**Joining:** At the bootstrap stage, the new joining node knows the ID and connection information (IP, port) of one other node, that it uses for bootstrapping. The new node initiates a lookup process for its own ID, the self lookup. During this process, the new node populates its \( k \)-buckets with other nodes in the network that are in the path between him and his bootstrap node.
Figure 3.4: Graphs depicting the number of connected components, and average degree centrality, after incremental node deletions, in a 10-regular graph, and Kademlia ($k = 5, 15, 20$) of 5000 (left side) and 15000 (right side) nodes.
In the following, we evaluate the resiliency and self-healing of OnionBots, and compare its network formation with Kademlia. Other works [77, 78] study the characteristics of Kademlia peer-to-peer network in more detail.

As we can see in Figures 3.4a and 3.4b, the self-repairing graph remains connected even when a large portion (90%-95%) of the nodes are deleted, compared to a normal graph (a graph with no self-repairing mechanism). Note that in a normal graph after 60% node deletion, the number of partitions increases sharply. We omitted plotting the number of connected components in a normal graph after 60% node deletion to provide more visible graphs (The number of connected component reaches 200 in 5000 node botnet and 600 in the 15000 node botnet). The Kademlia formation with \( K = 20 \), stays connected because of the large number of connections that nodes make. However, when we decrease \( k \) to 5 or 15, the network would start to disconnect. Please note that in Kademlia network even with \( k = 5 \), and \( k = 15 \), the nodes are still connected to a larger number of nodes (higher degree) than OnionBots.

As depicted in Figures 3.4c and 3.4d, the degree centrality of DDSR slightly increases with the node deletions. The increase is because the number of available nodes in the network is decreasing and given that DDSR maintains a constant bounded range degree. As a result, the degree centrality increase as the number of nodes decreases. Because of Kademlia network formation, nodes can have a larger variation in their degree. Therefore, each node deletion can fluctuate the degree centrality, based on its degree. As we can see, the Kademlia experiences a larger deviation and fluctuation in degree centrality, specially with larger values of \( k \).

3.4 Mitigation of Basic OnionBots

In this section, we look at different mitigation strategies against OnionBots. Mitigation and detection can take place at different levels, such as host level or network level. Host level remediation techniques include the use of anti-virus software and frameworks such as the Security Behavior Observatory (SBO) [79]. Because of the scaling limitation of such techniques, and the fact that the compromised hosts are rarely updated or patched, we focus on the network level strategies.
Many of the current detection and mitigation mechanisms are IP-based, and rely on the network traffic patterns or DNS queries to distinguish legitimate traffic from malicious traffic. However, current solutions do not work with OnionBots, since the Tor traffic is encrypted, non IP-based, and there are no conventional DNS queries to resolve the .onion addresses. Furthermore, even if an adversary captures a bot sample (e.g., by using honeypots or other similar techniques), and recovers the .onion address of its peers, the adversary is still unable to directly map these addressees to their corresponding IP addresses and take down the infected hosts. Since the proposed construction offers a self-repairing low-degree, low-diameter network, even after taking over a large portion of the bots, the botnet remains functional. Partitioning of a botnet network is particularly undesirable, since it makes the subgraphs disconnected. The capacity of the botnet decreases and the messages are not delivered to different partitions. As a result, each partition becomes a small botnet. The power of a botnet mostly depends on its large size and network capacity. An adversary needs to take down about 40% of the bots simultaneously, to even partition the network into two subgraphs, as depicted in Figure 3.5. Note that it means there is not enough time for the graph to self-repair. As we can see, the conventional solutions that ignore the privacy infrastructure construction of OnionBots are not effective. Therefore, we need to adapt our detection and mitigation methods, and integrate them into
the foundation of such infrastructures. In this section, we divide the network level mitigations into two categories; techniques that are generic to Tor, and schemes that are specific to OnionBots. In particular, we propose a new OnionBot specific mitigation method, called Sybil Onion Attack Protocol (SOAP).

### 3.4.1 Targeting OnionBots Through Privacy Infrastructures

Generic mitigations targeting Tor are based on denying access to the bots through the HSDirs. As described before, the list of HSDirs can be calculated by any entity who knows the .onion address (in case there is no descriptor-cookie). Hence, an adversary can inject her relay into the Tor network such that it becomes the relay responsible for storing the bot’s descriptors. Since the fingerprints of relays are calculated from their public keys, this translates into finding the right public key [22]. Nevertheless, it should be noted that an adversary needs to position herself at the right position in the ring at least 25 hours before (it takes 25 hours to get the HSDir flag). It is difficult to mitigate against many bots, since the adversary requires the computation power, and a prior knowledge of the .onion addresses. Furthermore, it disrupts the operation and user experience of the Tor network.

A more long term approach involves making changes to the Tor, such as use of CAPTACHAs, throttling entry guards, and reusing failed partial circuits, as described in [80]. Most of these approaches are targeted toward efficient handling of the increased requirement, rather than blocking or mitigating abuse. To provide a functional privacy infrastructure, it is of paramount importance for Tor to be able to handle the increased network bandwidth and load. Having said that, these mitigations are limited in their preventive power, open the door to censorship, can degrade Tor’s user experience, and are not effective against advanced botnets such as OnionBot. Please note with the next generation Tor hidden services, such mitigation would not be effective.

### 3.4.2 Sybil Onion Attack Protocol (SOAP)

We devised a mitigation mechanism that uses OnionBots’ very own capabilities (e.g., the decoupling of IP address and the host) against them. We first overview
Figure 3.6: SOAPing attack: node T is under attack by the compromised node C and its clones. At each step one of the clones initiates the peering process with the T, until it is contained. After several iterations, the network is partitioned, and the botnet is neutralized.
the attack here and then provide a step by step explanation as depicted in Figure 3.6. To attack a botnet and neutralize it, we first need to find the bots’ .onion addresses. This can be done either by detecting and reverse engineering an already infected host, or by using a set of honeypots. Although this is not a trivial task and requires a significant amount of effort, it allows us to infiltrate the botnet, traverse its network, and identify other bots. After identifying the bots’ .onion address, we run many hidden services, disclosing a subset of these as neighbors to each peer we encounter. Gradually over time, our clone nodes dominate the neighborhood of each bot and contain it. Note that we can run all of the clones on the same machine, because of the decoupling between the IP address and the host. Since there is no notion of trust or identity between the nodes in the botnet, we create many Sybil nodes that would infiltrate the botnet network formation and try to neutralize the network by diving the botnet into small, isolated, and disconnected islands. Given the small cost of such setup because of the IP decoupling, this technique is effective against onionbots.

Figure 3.6 depicts the SOAPing attack in different steps. Node $T$ is the target of the SOAPing attack, nodes $N_i$, are its neighboring bot nodes, and nodes $C$ are the adversary (e.g., the authorities) and his clones, which are represented with small black circles. In step 1, the botnet is operating normally, and none of $T$’s neighbors are compromised. In step 2, one of its peers, $N_4$, is compromised. Then, $N_4$ (now depicted as $C$), makes a set of clones (the small black circles). In step 3, a subset of $C$’s clones, start the peering process with $T$, and declare their degree to be a small random number, which changes to avoid detection (e.g., $d=2$). Doing so increases the chances of being accepted as a new peer, and replacing an existing peer of $T$. In step 4, $T$ forgets about one of its neighbors with the highest degree, $N_3$, and peers with one of the clones. The clones repeat this process until $T$ has no more benign neighbors (steps 5-8). As a result, $T$ is surrounded by clones and is contained (step 9). As we can see after many iterations, the adversary can partition the network into a set of contained nodes, and neutralize the botnet.

3.5 Beyond Basic OnionBots

As the authorities design mitigation techniques against advanced botnets such as OnionBots, the attackers will evolve their infrastructure to evade detection and
blocking. This arms race, results in the design and deployment of more sophisticated generation of OnionBots that limit the mitigation efforts. Such increased complexity, introduces new challenges, both for authorities and attacker. In the following, we briefly discuss a few of such challenges.

3.5.1 Overcoming SOAP

Although basic OnionBots are susceptible to the SOAPing attack, the attacker can mitigate against them, by using proof of work and rate limiting. In the proof of work scheme, each new node needs to do some work before being accepted as a peer of an already existing node. As more nodes request peering with a node, the complexity of the task is increased to give preference to the older nodes. The same approach can be used in the rate limiting, where the delay of accepting new nodes is increased proportional to the size of peer list. Although such actions increase the adversarial resilience of the network, they also decrease the flexibility and the recoverability of the network. Additionally, the attacker can make the NoN construction more resilient, and create a SuperOnionBot, by fully utilizing the decoupling provided by Tor.

3.5.2 SuperOnionBots

Even though basic OnionBots exploit the decoupling provided by Tor, they do not fully utilize the decoupling of physical host, IP address, and the .onion address. Note that a single physical host with one IP address can host many .onion addresses. By considering the aforementioned feature, we envision a new botnet formation called SuperOnion (SO). This construction consists of n SOs (physical hosts) and each SO simulates m virtual nodes, and each virtual node is connected to i other virtual nodes; a total of n * m virtual nodes, and m * i virtual peers per physical node. Although in this construct each virtual node is still susceptible to SOAPing attack, the physical hosting node is immune to such attacks, as long as one of its m virtual nodes is not SOAPed. Figure 3.7 illustrates a SuperOnion construction, with n = 5, m = 3, and i = 2.

To ensure the connectivity of the network, the hosting node periodically initiates a connectivity test by sending a probe message from each one of its m virtual nodes, and expecting to receive the message at its m – 1 other virtual
nodes. To maintain its stealth, the hosting node uses flooding techniques with low message complexity, such as gossiping. We make the assumption that the authorities are legally liable [81], and they cannot participate in the botnet activity. Since the messages are encrypted and indistinguishable, the authorities are not able to drop certain message and only allow the connectivity probe messages to pass through. If certain virtual nodes do not receive the messages, the host can deduce that they are SOAPed. After a physical node realized one of its virtual nodes is SOAPed, it creates a new virtual nodes, and initiates the bootstrapping stage, using peers from its currently connected virtual nodes.

3.6 Conclusion

Privacy infrastructures such as Tor had a tremendous impact on society, protecting users anonymity and rights to access information in the face of censorship. It also opened the door to abuse and illegal activities, such as ransomware [21], and a marketplace for drugs and contraband [82, 5]. In this chapter we envisioned OnionBots, and investigated the potential of subverting privacy infrastructures (e.g., Tor hidden services) for cyber attacks. We presented the design of a robust and stealthy botnet that lives symbiotically within these infrastructures to evade detection, measurement, scale estimation and observation. It is impossible for ISPs to effectively detect and mitigate such botnet, without blocking all Tor access. Additionally, OnionBots rely on a resilient self-healing network formation
that is simple to implement, yet it has desirable features such as low diameter and low degree. Such botnets are robust to partitioning, even if a large fraction of the bots are simultaneously taken down. In the scenario of a gradual take down of nodes, the network is also able to self-repair, even after up to 90% node deletions. More importantly, we developed SOAPing, a novel mitigation attack that neutralizes the OnionBots. We also suggested mitigation techniques that act at the Tor level.
Emergence of Tor as a popular tool and infrastructure in the last decade has attracted a large and varied user base. It is used today by millions of ordinary users to protect their privacy against corporations and governmental agencies, but also by activists, journalists, businesses, law enforcements and military [83].

The success and popularity of Tor makes it a prime target for adversaries as indicated by recent revelations. Despite its careful design that significantly improved users privacy against typical adversaries, Tor remains a practical system with a variety of limitations and design vulnerabilities, some of which were indeed exploited in the past [84, 22]. Due to the perceived security that Tor provides, its popularity, and potential implication on its users, it is important that the research community continues analyzing and strengthening its security.

This is specially important since users typically have a poor understanding of the privacy protection that Tor really provides, as evidenced by past events. For instance, in a highly publicized case, security researchers collected thousands of sensitive e-mails and passwords from the embassies of countries including India and Russia [85]. These embassies used Tor believing it provides end-to-end encryption, sending sensitive un-encrypted data through malicious exit nodes. Other research revealed that many users run BitTorrent over Tor, which is insecure and resulted in deanonymization [86]. Finally, recent incidents revealed that the Tor network is continuously being attacked by a variety of organizations from universities to governmental agencies, with difficult to predict ramifications [84].
Figure 4.1: Recent unexplained surge in the number of Hidden Services. The number of hidden services (.onion) suddenly tripled, before settling at twice the number before the surge.

Even more recently, the still unexplained sudden surge in the number of hidden services (.onion), more than tripling their number before returning to relatively smaller numbers (See Figure 4.1), indicates that the Tor network is not well understood, in part due to its peer-to-peer nature, the privacy services it provides that limit measurements, and the attacks that it attracts [87].

Tor’s security, by design, relies on the fact that a substantial number of its relays should not be malicious. It is however difficult to assess to what extent this condition holds true. The fact that many attacks are passive, makes it even harder to assess the significance of this threat. In this thesis, we developed a framework, techniques, and a system to provide some elements of the answer to this challenging problem.

We introduce the concept of honey onions (honions), to expose when a Tor relay with Hidden Service Directory (HSDir) capability has been modified to snoop into the hidden services that it currently hosts. We believe that such a
behavior is a clear indicator of sophisticated malicious activity, for it not only is explicitly undesired by the Tor Project [88] but also requires a modification to the Tor software, indicating some level of sophistication of the perpetrator.

Honions are hidden services that are created for the sole purpose of detecting snooping, and are not shared or publicized in any other form. Therefore, any visits on the server side of the honion is a clear indication that one of the HSDirs that hosted it, is snooping. Since hidden services are hosted on multiple HSDirs and change location on a daily basis, it is not easy to infer which HSDir is the malicious one (we use the terms malicious and snooping interchangeably). The visits information leads to a bipartite graph connecting honions and HSDirs. Finding the smallest subset of HSDirs that can explain honion visits provides a lower-bound on the number of malicious HSDirs. This has the benefit of giving a sense of the scale of malicious behavior among Tor relays. We show that this problem can be formulated as a Set Cover, an NP-Complete problem. We develop an approximation algorithm to this specific problem as well as an Integer Linear Program (ILP) formulation. We build a system to deploy the honions along with a schedule for the lifetime of each one of them to maximize the collected information without generating an excessive number of hidden services. The generated honions have a lifetime of one day, one week, or one month.

Throughout the experiment, which lasted 72 days, the maximum number of generated honions did not exceed 4500 hidden services (which is significantly lower than the anomaly that hidden services were experiencing). Based on the experimental data, we are able to infer that there are at least 110 snooping HSDirs. A careful analysis of the experimental data and results from the ILP solution, allows us to infer most of the misbehaving HSDirs and their most likely geographical origin. Based on these results we are able to classify misbehaving HSDirs in two main categories, immediate snoopers, and delayed snoopers. Immediately and deterministically visiting a honion results in a higher detection and identification. However, delaying and randomization reduces the traceability (as other HSDir who hosted the honion could also be blamed) at the expenses of potentially missing key information that the hidden service creator might put for only a short period of time. Therefore, a smart HSDir snooper has to trade-off delay (and risk of missing information) with risk of detection. In this paper, we discuss the behavior and characteristics of the malicious HSDirs. We found out that more than half the malicious HSDirs are of the delayed type, and are hosted on cloud infrastructure. Our contributions can be summarized as follows:
• The honey onion framework for detecting snooping HSDirs.
• An approximation algorithm and Integer Linear Program for estimating and identifying the most likely snooping HSDirs.
• An experimental study leading to the discovery of at least 110 snooping HSDirs and a peek into their behavior.

We discuss related work in Section 4.1, followed by outlines of our approach, and system architecture in Section 4.2. Section 4.3 provides the formalization of the detection and identification problem, shows the reduction to the set cover problem, and the approximation algorithm as well as the Integer Linear Programming formulation. In Section 4.4, we discuss our implementation of the system, report on the experimental results when processed by the identification algorithms. In Section 4.5, we discuss the experimental results and the characteristics and behavior of malicious HSDirs.

4.1 Related Work

We are the first to study the snooping HSDirs and malicious actor inside the hidden service and darkweb. Previous work studied other aspects of Tor’s security [89] such as malicious traffic, de-anonymization attacks, and misbehaving relays such as Exit nodes. For example, Kwon et al. [10], investigated weaknesses in the design of hidden services that allows them to break the anonymity of hidden service clients and operators passively. The authors show that paths established through the Tor network, used to communicate with hidden services exhibit a different behavior compared to a general circuit. Jansen et al. [90], introduce the sniper attack, a low cost denial of service attack against Tor that an adversary may use to anonymously disable Tor relays. The attack utilizes valid protocol messages to boundlessly consume memory by exploiting Tor’s end-to-end reliable data transport. Sun et al. [91], demonstrate the Raptor attack that can be launched by Autonomous Systems (ASes) to compromise user anonymity. In the first step, AS-level adversaries exploit the asymmetric nature of Internet routing to increase the chance of observing at least one direction of user traffic. Second, AS-level adversaries would exploit churn in Internet routing to fake BGP paths for more users. Third, strategic adversaries manipulate Internet routing via BGP hijacks and interceptions for traffic analysis.
Our work focuses on detection and classification of misbehaving hidden services directories (HSDirs), an essential component of the hidden services architecture and the privacy of users. A previous work by Winter et al. [2] exposes malicious exit nodes. The authors developed two exit relay scanners, one for credential sniffing and one for active man-in-the-middle (MITM) attacks. The modules aimed for detecting common attacks, and were used to probe all exit relays over a period of months. The authors showed 65 malicious or mis-configured exit relays are engaging in a multitude of different attacks. Furthermore, a fraction of all attacks were coordinated rather than isolated. In another work [92], the authors propose sybilhunter, a technique to detect Sybil relays based on their appearance and characteristics such as configuration, fingerprint, and uptime sequence using the consensus document. The malicious and sybil relays can be used for tampering with exit traffic, website fingerprinting, bridge address harvesting, and end-to-end correlation attacks.

Another group of work look at the content and popularity of hidden services and malicious traffic inside Tor network. For example, Ling et al. [93] present TorWard, a system for the discovery and the systematic study of malicious traffic over Tor. The system allows investigations to be carried out in sensitive environments such as a university campus, and allows to avoid legal and administrative complaints. The authors investigate the performance and effectiveness of TorWard by performing experiments and showing that approximately 10% of Tor traffic can trigger IDS alert. Biryukovhs et al. [82] collected 39824 hidden services descriptors and scanned them for open ports, and in the case of HTTP services, analysed and classified their content. They studied popularity of hidden services by looking at the request rate for hidden service descriptors by clients. The authors findings reveal that while the content of Tor hidden services is rather varied, the majority of hidden services belong to botnets, followed by adult content and drug markets.

Other works investigated flaws in Tor’s design that allows opportunistic de-anonymization of hidden services. Biryukov et al., [22], discover and exploit a flaw in the design and implementation of hidden services in Tor, which allows an adversary to measure the popularity of any hidden service, block access to hidden services, and ultimately deanonymize hidden services. Furthermore, the authors investigated a case of a botnet using Tor hidden services for command and control channels; Silk Road, a hidden service used to sell drugs and other contraband; The hidden service version of the DuckDuckGo search engine. Other work [94], document their findings on probing the network topology and connectivity of Tor.
relays. Given that Tor relies on the hardness of linking a user’s entry and exit
nodes, the authors demonstrate how the leakage of the Tor network topology can
be used in attacks to traceback from an exit node to a small set of possible entry
nodes. Therefore, defeating the anonymity of the users in Tor.

Another direction of research on hidden services is the study of leakage of
onion services, due to design and implementation flaws or misconfiguration. For
example, even though Tor is designed to prevent leakage of onion addresses into
the DNS system, many request are still observed. Thomas and Mohaisen [95]
measures the leakage of onion addresses at the root DNS servers (A and J), and
provide the popularity of different hidden services categories based on the leaked
requests. Another work [96] tries to identify location leaks in hidden services sen-
sitive information in the content served by the hidden service or its configuration
that discloses the server’s IP address. This work does not rely on flaws of the
Tor protocol, but relies on misconfiguration of the servers. CARONTE visits the
hidden service, extracts Internet endpoints and looks up unique strings from the
hidden service’s content. This is followed by examination of the hidden service’s
certificate chain to extract candidate Internet endpoints where the hidden service
could be hosted. Finally, it validates those candidates by connecting to them.

4.2 Approach

In the following section, we overview the approach and the architecture of our
honey onion detection platform. The steps of flow of actions are depicted in
Figure 4.2.

4.2.1 HOnions Generation

In order to automate the process of generating and deploying honions in a way
that they cover a significant fraction of HSDirs, we developed several scripts. The
scripts create configuration files for Tor relays, called torrc. In particular, the
torrc file specifies the SOCKS port, the hidden service directory to store and
read the private key, the advertised port of hidden service, and the port where a
server is running on the localhost as described in the next subsection. Since the
majority of hidden services are run as HTTP servers [82], we also advertised our
Figure 4.2: Flow diagram of the honion system. We generate a set of honions to cover all the HSDirs and run a server behind each one. Here, we only show one descriptor per honion. When a visit happens to one of the honions, we can infer which HSDirs hosted it (and knew about its existence) using the consensus document and the list of relays. After identifying the potential suspicious HSDirs, we add the candidates to the bipartite graph.

servers as HTTP provider. This would allow to attract the attention of snoopers, without raising suspicion.

A key constraint in this process was to minimize the number of deployed honions. This derives primarily from our desire to not impact the Tor statistics about hidden services (specially given the recent surge anomaly). Secondarily, given that behind each honion there should be a running process to serve the pages and to log the visits, we are practically limited by our infrastructure hardware/server capabilities. We now discuss the process that allowed us to determine how many honions should be generated to cover at least 95% of the HSDir for every batch.

If each honion was only placed on a single random HSDir, the probability for each HSDir to host a honion is $p_0 = \frac{1}{N_{hdirs}}$, where $N_{hdirs}$ is the number of HSDirs. The number of HSDirs is publicly available and can be found on the Tor’s official metrics. Since there are two descriptors, derived independently, this is equivalent to doubling the number of honions ($m$). Since each descriptor is placed on a set of three adjacent HSDirs, the probability of a descriptor being hosted on an HSDir is approximated by $p \approx 3p_0 = \frac{3}{N_{hdirs}}$. After generating $m$ honions, the probability that an HSDir is not covered by the $2m$ descriptors is approximated $(1 - p)^{2m}$. To cover a fraction $f$ of HSDir, we need:
\[ f = 1 - (1 - \frac{3}{N_{hstdirs}})^{2m} \]

This implies that the necessary number of honions to be generated should be as follows:

\[ m = \frac{\log(1 - f)}{2 \times \log(1 - (1 - \frac{3}{N_{hstdirs}}))} \]

Using this formula and considering that the number of HSDirs \( N_{hstdirs} \) is approximately 3000, we could infer that we need to generate 1497 (rounded to 1500) honions to cover all HSDirs with 0.95 probability. We used 1500 honions per batch (daily, weekly, or monthly) and could verify that 95% of the HSDirs were systematically covered, therefore, validating our approximation.

An alternative approach would have been to generate a very large number of honions or interactively generating them until all HSDirs are covered. However, both approaches have drawbacks and limitations. For instance, to iteratively cover the HSDirs, one needs to have a perfect synchronization between the generation process and Tor consensus documents. Given that the consensus document is updated frequently during the day and depending on HSDirs and the client relays view of the network, the results can be different. As for generating a large number of honions, it can overload the Tor network, disturb its statistics primitives, and also requires us to run an excessive number of server processes.

### 4.2.2 HOnion back end servers

Each honion corresponds to a server process/program that is running locally, attached to unprivileged ports. The server programs are later announced as HTTP servers to the Tor network. The server behind hidden services, should not be running on a public IP address. Otherwise it can be detected and deanonymized by exploiting its unique strings and other leakages. This has become relatively
easy, given the availability of databases of the whole Internet scans. To avoid leaking information we return an empty page for all the services. It does not allow an adversary to draw any conclusions about the hosting server. We initially considered using fake pages mimicking real typical hidden services websites. However, similarities between pages might alert an adversary about the existence of a honeypot/honey onion.

4.2.3 HOnions generation and deployment schedule

To keep the total number of honions small, we decided on three schedules for the generation and placement of the honions, daily, weekly, and monthly. The three schedules allow us to detect the malicious HSDirs who visit the honions shortly (less than 24 hours) after hosting them. Since the HSDirs for hidden services change periodically, more sophisticated snoopers may wait for a longer duration of time, so they can evade detection and frame other HSDirs. The daily schedule would miss such snoopers; therefore, we defer to the weekly and monthly honions to spot such adversaries. Imagine there is a visit on weekly or monthly honions, while there is no visits to the daily honions. Since all honions are running simultaneously, and all HSDirs are hosting honions in all three schedules, this indicates that some malicious HSDirs are delaying their snooping. For the adversary, this is a trade-off between accuracy and stealthiness. This is because some hidden services may have a short life span and would be missed by the snooping HSDir if he waits too long. Note that the lifespan of hidden service is not known.

4.2.4 Logging HOnions visits

We log all the requests that are made to the server programs and the time of each visit. We also log the headers that are sent with HTTP requests, including the User-Agent field. The time of a visit allows us to determine the HSDirs that have hosted any specific honion. Recording the content of the requests allows us to investigate the behavior of the snoopers. Since we advertise our servers on port 80, we can investigate the request types and content that are made by snoopers. Furthermore, we can detect automated headless crawls as opposed to the requests made by browsers (e.g., Tor browser). This is because since they make request
for extra elements such as the small icon that is shown in the browser near the URL address bar (i.e., favicon.ico).

4.2.5 Identifying snooping HSDirs

Based on the visited hidden server, the time of the visit, and the HSDir that have been hosting the specific onion address prior to the visit, we can mark the potential malicious and misbehaving HSDirs. If the visit has happened after an HSDir has hosted a specific onion address, that HSDir is a candidate. Then we add the candidates to a bipartite graph, which consists of edges between HSDirs and the visited honions, as further described in section 4.3. The analysis of this graph allows us to infer a lower bound on the number of malicious HSDirs as well as the most likely snoopers. We need to find the lowest number of HSDirs that can explain all the visit (edges) in our bipartite graph.

4.3 Estimation & Identification of Snooping HS-Dirs

In order to formally reason about the problem of identifying malicious HSDirs, we first introduce a formal model and notation for the Honey Onions system. First, $HO$ denotes the set of honey onions generated by the system that were visited, and $HSD$ the set of Tor relays with the HSDir flag (so far referred to as HSDir relays). The visits of honions allow us to build a graph $G = (V,E)$ whose vertices are the union of $HO$ and $HSD$ and edges connect a honion $ho_j$ and HSDir $d_i$ iff $ho_j$ was placed on $d_i$ and subsequently experienced a visit. $G$ is by construction a bipartite graph.

$$HSD = \{d_i : \text{Tor relays with HSDir flag}\}$$

$$HO = \{ho_j : \text{Honey Onion that was visited}\}$$

$$V = HSD \cup HO$$

$$E = \{(ho_j, d_i) \in HO \times HSD | ho_j \text{ was placed on } d_i \text{ and subsequently visited}\}$$
We also note that each honion periodically changes descriptors and therefore HSDirs (approximately once a day). However, an HSDir currently a honion ho, cannot explain visits during past days. Therefore, each time a honion changes HSDirs, we clone its vertex ho to ho′, and only add edges between ho′ and the HSDirs who know about its existence when the visit happened.

4.3.1 Estimating the number of snooping HSDirs

Since each honion is simultaneously placed on multiple HSDirs, the problem of identifying which ones are malicious is not trivial. We first formulate the problem of deriving a lower-bound on their number by finding the smallest subset $S$ of $HSD$ that can explain all the visits (meaning that for each visited honion, there is a member of $S$ who knew about its existence and could therefore explain the visit). The $S$ is therefore a solution to the following problem:

$$\arg\min_{S \subseteq HSD} |S : \forall (ho_j, d_i) \in E, \exists d'_i \in S \land ((ho_j, d'_i) \in E)|$$ (4.1)

The size $s$ of the minimal set tells us that there cannot be less than $s$ malicious HSDirs who would explain the visits. Furthermore, when $s$ is relatively small compared to $N_{hstdirs}$, any HSDir identified as an explanation of multiple visits is highly likely to be malicious. This derives from the fact that the probability of co-hosting a honion with a malicious HSDir, decreases exponentially as a function of number of visits.

4.3.2 Reduction from set cover

Finding the smallest set $S$ as defined by Equation 4.1, is not trivial as one can easily see that it is equivalent to the hitting set problem, which itself is equivalent to the set cover problem. The set cover problem is well known to be NP-Complete. An intuitive sketch of proof for the equivalence to set cover is as follows. For each HSDir $d_j$ define the set of honions $O_j = \{ho_i | (ho_i, d_j) \in E\}$. Solving Equation 4.1 amounts to finding the smallest set of $O_j$ that covers all the visited honions. The set cover problem has an $\ln(n) + 1$ approximation algorithm where $n$ is the size of the set to be covered [97]. Based on this, we derive the following heuristic, with $\ln(|HO|) + 1$ approximation ratio. The advantages of this heuristic is its
low computation complexity $O(|E|)$.

\begin{algorithm}
\textbf{Input:} $G(V, E)$: Bipartite graph of honions to HSDirs  \\
\textbf{Output:} $S$: Set explaining visits  \\
1 $S \leftarrow \emptyset$  \\
2 while $V \cap HO \neq \emptyset$ do  \\
3 \hspace{1em} Pick $d \in V \cap HSD$ : with highest degree  \\
4 \hspace{1em} $V \leftarrow V \setminus \{d\ \text{and its honion neighbors}\}$  \\
end

\textbf{Algorithm 4:} Minimal HSDir greedy approach

### 4.3.3 Formulation as an Integer Linear Program

Solving the problem defined by Equation 4.1, can also be formulated as an Integer Linear Program (ILP). Integer linear programming is an optimization algorithm for linear functions, where variables are integers. Let $x_{1 \leq j \leq |HSD|}$ be binary variables taking values 0 or 1. Solving Equation 4.1, consists of finding Integer assignments to the $x_j$ such that:

$$\min_{x_1, \ldots, x_{|HSD|}} \sum_{j=1}^{|HSD|} x_j$$

subject to $\forall ho_i \in HO$ $\sum_{\forall j : (ho_i, d_j) \in E} x_j \geq 1$

While this ILP will give the optimal solution, it has exponential computation complexity in the worst case. In a subsequent section, our experimental results show that although ILP performs fairly well for our setup, it is significantly slower than the heuristic approach.

### 4.3.4 Ground Truth & Simulations

Since we do not have access to the ground truth for snooping HSDirs, we use simulations to evaluate the accuracy of our ILP detection technique. We simulated a set of snooping HSDirs and used ILP technique to detect them. These simulations provide us with a level of confidence with our most likely node detection. We gradually increase the percentage of snoopers to measure its impact on the accuracy of results. Figure 4.3 depicts the simulation results. As we can
Figure 4.3: ILP detection accuracy using simulation and the ground truth. Detection accuracy is beyond 80% as long as the number of snooping HSDirs is less than 10%. The larger the set of snooping HSDirs, it is harder to detect the right snooper. The detection accuracy is above 95% as long as less than 4% of the HSDirs are malicious.

see, by increasing the percentage of the snoopers the accuracy of the results decreases. For example, as long as the percentage of snoopers is below 3% of the HSDirs, our results are more than 95% accurate. Even when the percentage of the snoopers increases to 10%, we can detect the snoopers with 80% accuracy on average. Note that by increasing the number of malicious nodes inside the Tor network, other security and privacy services can also be compromised.

### 4.4 Detection Infrastructure & Results

In this section, we discuss the implementation and deployment of the detection infrastructure as highlighted in Section 4.2 and depicted in Figure 4.2.
4.4.1 Implementation and Deployment of the Detection Platform

We developed simple HTTP servers to listen on specific ports for incoming requests. Upon receiving a request, each server would log the time and full request and headers into separate files. At first, we developed the HTTP servers using Python and Flask web framework. However, because of the size that is occupied by the framework and the interpreter, we faced difficulties in scaling our detection platform. The programs when instantiated in memory would take up to 40 MB, including the shared libraries. Running 1500 instances would take up to 12GB. Meaning each instance on average could take about 8-9 MB. As a result, we decided to port the code to C, without using any external third party library or framework. We relied solely on the BSD Sockets API. This allowed us to reduce the size of the code, including the shared libraries, to 6 MB. Running 1500 instances with the ported code only occupied around 2GB, meaning each instance, on average, occupied less than 1.5 MB, therefore, reducing the resource allocations by 6 times.

We distributed the 1500 honions over 30 Tor relays equally, to avoid overloading a single relay and reducing performance and responsiveness of the hidden services. We created scripts that would automatically generate and place new honions based on the three schedules discussed earlier (daily, weekly, monthly). Each schedule was running on a separate Virtual Machine to isolate the infrastructures.

4.4.2 Analyses of the Results and Observations

We started the daily honions on Feb 12, 2016; the weekly and monthly experiments on February 21, 2016, which lasted until April 24, 2016. During this period there were three spikes in the number of hidden services, with one spike more than tripling the average number of hidden services (Figure 4.1). First spike was on February 17, second on March 1 (the largest), and the last on March 10. These spikes attracted a lot of attention from the media [98]. However, there is still no concrete explanation for this sharp influx of hidden services. There are some theories suggesting that this was because of botnets, ransomware, or the success of the anonymous chat service, called Ricochet. However, none of these explanations can definitely justify the current number of hidden services.
Figure 4.4: Plot of the visits to the honions. The daily onions show snooping HS-Dirs, before the “mystery” spike in hidden addresses. The number and intensity of the visits is increased after the spikes.
Table 4.1: Top 5 cloud providers of the most likely snooping HSDirs.

<table>
<thead>
<tr>
<th>Provider</th>
<th>HSDirs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alibaba California</td>
<td>15</td>
</tr>
<tr>
<td>Digital Ocean</td>
<td>7</td>
</tr>
<tr>
<td>Online S.A.S.</td>
<td>7</td>
</tr>
<tr>
<td>OVH SAS</td>
<td>6</td>
</tr>
<tr>
<td>Hetzner Online GmbH</td>
<td>6</td>
</tr>
</tbody>
</table>

Snooping HSDirs Nature. In total we detected, at least, 110 malicious HSDir using the ILP algorithm (the ILP took about 2 hours), and about 40000 visits. More than 70% of these HSDirs are hosted on Cloud infrastructure. Around 25% are exit nodes as compared to the average, 15% of all relays in 2016, that have both the HSDir and the Exit flags. Furthermore, 20% of the misbehaving HSDirs are, both exit nodes and are hosted on Cloud systems. The top 5 cloud providers are Alibaba-California (15 detected HSDirs), Digital Ocean (7), Online S.A.S. (7), OVH SAS (6), and Hetzner Online GmbH (6). Hosting the relays on the cloud provides a level of anonymity and protection for the operators. Specially, some of the cloud providers reside in European countries with strong privacy protection laws that help to hide the identity of the real owners of the relays. Moreover, some of the providers accept bitcoin payments, which provides a greater anonymity for the relay operators. Table 4.1 summarizes the cloud providers and the number of malicious HSDirs each one is hosting.
Table 4.2: Type of the snooping HSDirs. More than 70% are hosted on Cloud.

<table>
<thead>
<tr>
<th>Cloud</th>
<th>Exit</th>
<th>Cloud &amp; Exit</th>
<th>Not Cloud &amp; Not Exit</th>
</tr>
</thead>
<tbody>
<tr>
<td>81</td>
<td>27</td>
<td>23</td>
<td>25</td>
</tr>
</tbody>
</table>

Alibaba cloud belongs to Alibaba Group, the Chinese e-commerce company, with servers in the US, Europe and Asia. All instances that we spotted were hosted on the US West Coast data centers. Digital Ocean is an American cloud provider that targets software developers, located in New York. Online S.A.S and Hetzner Online GmbH are two German cloud provider, and OVH SAS is another European cloud provider, located in France.

Exit nodes play a significant and sensitive role in the Tor platform, and can cause legal problems for their operators [99]. At the same time it is known that some Exit nodes are not benign and actively interfere with users’ traffic. There is a Bad Exit flag, to warn users not to use these relays as exit nodes. None of the exit nodes that we identified have been identified as Bad Exit nodes. This can be because they do not perform active MITM attacks, and evade detection. Table 4.2 summarizes the type of the HSDir relays. As shown by previous work [100] correlation attacks can be performed by using the client request time at HSDirs and correlating it with the circuit establishment of a client. Furthermore, if the adversary is controlling the Guard nodes or the Rendezvous point it allows him to de-anonymize the clients and hidden services.

Figure 4.5 illustrates a typical bipartite graph of a daily visit. The black nodes indicate the malicious HSDirs marked by ILP. The gray nodes are the honions that have been visited, and the colored nodes are all other HSDirs that have hosted the honions. Note that many of the honions belong to a separate component in the graph, and in one connected component more than one HSDir is suspicious. As we can see, in larger graphs the number of “suspicious” relays in much larger. Given that each visited honion adds 6 suspicious relays to the set, the ILP algorithm can find the lowest number of HSDirs that can explain all the visits. Please note that the answer is not unique, but it still provides a lower bound.

Snooping HSDirs Geolocation. Figure 4.6 depicts the most likely geolocation and type of the misbehaving HSDirs. In the interest of space, we have omitted the only HSDir in Australia. The black icons represent the HSDirs hosted on a cloud platform that are exit nodes as well. The Red icons represent the nodes
hosted on cloud that are not exit nodes, the blue icons represent the exit nodes that are not hosted on the cloud, and the green icons are the relays that are neither exit nodes, nor hosted on the cloud. Our results indicate that there are no snooping HSDirs in China, Middle East, or Africa. It is not surprising since in these regions and countries Tor is heavily blocked [101]. Furthermore, more than 70% of the snooping HSDirs are hosted on Cloud systems, and many of the cloud providers’ data centers are located in Europe and Northern America. Table 4.3 summarizes the top 5 countries where the malicious HSDirs are located.

Note that 15 of the 37 HSDir in the USA, belong to Alibaba cloud data centers, followed by Digital Ocean and Linode. The gelocation of the snoopers is a representative of the Tor users as well. Northern American and Europe are the largest users of Tor. According to Tor Metrics, 40% of estimated users are form Germany and USA.

Classifying the Behavior and Intensity of the Visits. Most of the visits were just querying the root path of the server and were automated. However, we identified less than 20 possible manual probing, because of a query for favicon.ico, the little icon that is shown in the browser, which the Tor browser requests. Some snoopers kept probing for more information even when we returned an empty page. For example, we had queries for description.json, which is a proposal for all HTTP servers inside Tor network to allow hidden services search engines such as Ahmia, to index websites. Ahmia is a hidden services search engine which probes the description.json file for better classification of
<table>
<thead>
<tr>
<th>USA</th>
<th>Germany</th>
<th>France</th>
<th>UK</th>
<th>Netherlands</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>19</td>
<td>14</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4.3: List of top 5 countries with the most likely misbehaving HSDirs. USA and Germany are home to 40% of Tor users worldwide.

Figure 4.6: The global map of detected misbehaving HSDirs and their most likely geographic origin (in the interest of space we have omitted the only HSDir in Australia). The black icons represent the HSDirs hosted on a cloud platform that are exit nodes as well. The Red icons represent the nodes hosted on cloud that are not exit nodes, the blue icons represent the exit nodes that are not hosted on the cloud, and the green icons are the relays that are neither exit nodes, nor hosted on the cloud.

hidden services and providing more accurate results to search queries. A typical `description.json` includes information such as `title`, `description`, `keywords`, and `language`.

We identified a small number of well-behaving civilized crawlers, asking for `robots.txt` and `sitemap`. The robots exclusion standard (`robots.txt`) informs the web crawler bot about areas of the website that should not be processed or scanned. Sitemaps, inform the crawlers about URLs on a website that are available for crawling. One of the snooping HSDirs (5.*.*.*:9011) was actively querying the server every 1 hour asking for a `server-status` page of Apache. It is part of the functionality provided by `mod_status` in Apache, which provides information on server activity and performance. This can be an indication of the adversaries’ effort for reconnaissance and finding vulnerabilities and generally more information about the platform. Tools such as onionscan [102] look for such characteristic to ensure attackers cannot easily exploit and deanonymize
hidden services, because of an oversight in the configuration of the services.

We detected different attack vectors, such as SQL injection, targeting the `information_schema.tables`, username enumeration in Drupal (`admin/views/ajax/autocomplete/user/`), cross-site scripting (XSS), path traversal (looking for `boot.ini` and `/etc/passwd`), targeting Ruby on Rails framework (`rails/info/properties`), and PHP Easter Eggs (`?=PHP*-.*-*`). The normal web attack surface is applicable and prevalent in case of darkweb and hidden services.

In general, the snoopers showed a wide range of behavior. Some only appeared after the first spike in the number of hidden services and disappeared afterwards and gone offline (Figures 4.7b & 4.7c), while some of them came back after a month (Figure 4.7d). On the other hand, one snooper changed its behavior and turned into a snooping HSDirs after a while (Figure 4.7a). Such behavior
can be used for further classification of snoopers in terms of their motives and characteristics. For example, some can be researchers in the field, while another group can be attackers that seek financial gains from compromising the hosts or hidden services. Previous work have looked into the content and popularity of hidden services, that can be viewed as intrusive. As a result the Tor community established a Tor Safety Board to make sure that research conducted on Tor network is safe and secure for users.

4.5 Discussion & Future Work

Based on our observations, not all snooping HSDirs operate with the same level of sophistication [103]. While some do not visit the hosted honions immediately and therefore evade detection through daily honions, our weekly and monthly honions can detect them. We believe that behavior of the snoopers can be modeled and categorized into four groups: (1) Persistent-Immediate snoopers, where they immediately (within a day) and systematically probe all .onion addresses they service; (2) Persistent-Delayed, where they systematically probe all .onion addresses they service but with a fixed delay \(d\); (3) Randomized with Deterministic Delay, where they probe a learned .onion address with probability \(p\) after \(d\) days; (4) Probabilistic Snoopers, where once they learn about .onion addresses, they probe, after \(d\) days, according to distribution function \(p(d)\). Further work is needed to define more models and develop techniques to detect and identify the more sophisticated snoopers. Such categorization can also be used to study the motivations of snoopers. For example, security and privacy researcher working on hidden services and dark web, as opposed to attackers who seek financial gains by compromising host and hidden services, or state level actors.

Since some HSDirs probe deep in the hidden services, by using vulnerability discovery and automated attack tools, it would be interesting to create pages with login forms or more enticing content to engage the snoopers. However, one should carefully consider the legal, ethical, and privacy aspects of such investigations and studies.

The rise and popularity of cloud services allow entities to provision infrastructures without much overhead, which makes it difficult to detect malicious Tor nodes. In this competitive market, many cloud providers try to distinguish
themselves by providing more privacy and anonymity for their clients. For example, flokinet.is, advertises its services as a platform suitable for freedom of speech, investigative journalism, and perfect for whistleblowers, with servers in Romania, Finland, and Iceland. Although, one cannot deny the benefit of such privacy infrastructures, they can also be subverted and misused for malicious and harmful activities. Furthermore, cloud providers such as Vultr, even accepts payments in the form of bitcoins, which prevents the traceback and identification of misbehaving entities.

It is noteworthy that we continued the deployment of the honions, and after making our work public [104, 105], we observed a new trend of snooping behavior. The snoopers delay their visits to avoid identification, which indicates that the misbehaving HSDirs have already adapted their techniques. Figure 4.8 depicts the new trend of visits, where snoopers are becoming more sophisticated and delay their visits. Note that we count multiple visits to the same honion within one day, only once for this graph. We also discussed our work with some of the Tor Project people and learned that they have been aware of the problem and developed techniques (although different from ours) to identify and block misbehaving HSDir relays. Furthermore, they are also working on a new design
to mitigate various attacks against hidden services [13]. The new design will hide the hidden services address by encrypting the address by a key that is derived from the hidden service address. This technique ensures that the undisclosed hidden services address will not be leaked. However, the new design still needs to be investigated and audited for potential security and privacy vulnerabilities and risks.

Another direction is to explore the capabilities of Intel SGX [106, 107], and make modifications to Tor to run inside enclaves. For example, Kim et al. [108], adapt the Tor code to run inside the SGX platform to prevent code modification and limit the information exposed to untrusted parties. However, such approach introduces performance overhead, and limits its adaptation to many real world scenarios; specially use cases with high performance and low latency requirements.

4.6 Conclusion

Tor is a widely popular system for protecting users anonymity. However, at its core it relies on the non-malicious behavior of its peer-to-peer nodes. In this chapter, we introduced honey onions, a framework for methodically estimating and identifying the most likely Tor HSDir nodes that are snooping on hidden services they are hosting. We proposed algorithms to both estimate the number of snooping HSDirs and identify them. Our experimental results indicate that during the period of our work (72 days), at least 110 such nodes were snooping information about hidden services they hosted. We reveal that more than half of them were hosted on cloud infrastructure and delayed the use of the learned information to prevent easy traceback. Another interesting finding is that although a large number of snooping HSDirs were hosted on US IP addresses (37), several (15) were actually hosted on Alibaba’s data center in California. Furthermore, we revealed that the snoopers are adapting their techniques in the light of our work and finding, to mitigate identification.
Chapter 5

Privacy Preserving Study of Hidden Services Lifespan

Data collection and study of networked systems is difficult, because of two aspects: technical and ethical. The collected raw data, or even the processed data after analysis, can reveal information about the clients and users of the systems. Even some attempts to anonymize the collected or published data have failed to protect users’ privacy and have resulted in de-anonymization of individuals [109].

Privacy infrastructures such as Tor have even higher privacy risks, due to the nature of assumptions that are made, and inherent privacy dynamics of such network. For example, Tor’s security and privacy guarantees rely on the assumption that the majority of the relays are well behaving and that the relays do not carry out network correlation attacks. Any study that uses the data collected from the relays, needs to consider the implications of data collection, storage, and analysis. Furthermore, such a work should consider the impact of the final published output to ensure such information would not introduce new risks to the users. At the same time, the relays that are collecting data become a more attractive target of attacks for stealing the data, or coercion of the operators to release the collected data. Therefore, the collected data needs to be resilient to such adversarial models.

Currently, not much is known about the hidden services nature, and longevity. Studying the lifespan\(^1\) of hidden services can provide manifold benefits. For exam-

\(^1\)we use the terms lifespan and longevity interchangeably.
ple, it allows investigation of the maliciousness of domains based on their lifespan. Short-lived hidden services are more likely not to be legitimate domains, as compared to long-lived domains. Furthermore, such knowledge provides insights into the performance and resource allocation requirements of privacy infrastructures. Such analysis combined with the popularity measure of hidden services, provides knowledge into the interworkings of the dark web. The distributed nature of Tor and hidden services makes such study non-trivial and introduces challenges that need to be carefully addressed. Along the lines of previous privacy-preserving studies on Tor, we design a set of techniques that compute an estimate of the distribution of the lifetime of onion services without enabling any single party to learn anything more than their share if the other parties abort.

The study of longevity of hidden services is trivial if one has access to all HSDirs in the network, since it translates to counting the number of days a hidden service descriptor is uploaded to the HSDirs. However, given the volunteer run nature of Tor relays, it is not practical. In this work, we investigate the longevity of hidden services by collecting data only from a small subset of the HSDir relays (2%). We develop simulation and extrapolation techniques to infer the lifespan of hidden services, using the collected data from a small number of relays, with high accuracy. Furthermore, we show the feasibility of meaningful study of hidden services statistics in a privacy preserving manner that adheres to the Tor’s standards, and ethical and safety requirements. Previous work looking at hidden services [3], did not consider any safeguards, and their results were based on the probing of the collected onion address from their HSDirs, which is clearly against Tor user agreement. In contrast, our work does not store any information about the onion services. Furthermore, we do not perform any probing of the hidden services.

The rest of this chapter is arranged as the following. We first review the related work in Section 5.1, followed by a discussion of Tor’s safety board and our procedure to get approval in Section 5.2. In Section 5.3, we overview the privacy preserving data collection protocol, followed by our extrapolation techniques in Section 5.4. In Section 5.5, we outline the deployment and implementation details of our infrastructure, followed by the measurement results in Section 5.6. We discuss possible techniques and approaches for protection against dishonest adversaries in Section 5.7, followed by alternative approaches to secure computation in Section 5.8. We finally conclude in Section 5.9.
5.1 Related Work

In this section, we review the related work on study of Tor hidden services, and advancements in the privacy preserving data collection and analytics.

One of the first studies on the Tor network by McCoy et al. [110], collected data from the network by participating and contributing relays to the Tor network. The authors were studying three main questions: “How is Tor being used?”, “How is Tor being mis-used?”, and “Who is using Tor?”. Although such research questions and their answers are important, unfortunately, the authors did not consider required safeguards to protect the privacy of the users, and mitigate against its compromise. Another example of studies looking at Tor and hidden services that does not implement security and privacy safeguards is [3], where authors collect information about hidden services by setting up HSDirs. Additionally, the authors actively probe the collected hidden services for content. Such work, have been widely criticized by other researchers [111] and the Tor Project. As a result tools such as ExperimenTor [112] and Shadow [113] were introduced, for researchers to experiment with simulated Tor network. As a measure to prevent future unethical work, the Tor Research Safety Board [114] was established to review the research projects studying the Tor network, to ensure such works do not harm users’ privacy. In the next section we will further discuss the Tor Research Safety Board.

One of the early works on privacy preserving data collection and study of the Tor network, specially hidden services, is a work by the Tor team [115] to collect and count the number of unique hidden services. The authors use differential privacy, controlled noise addition to the statistics, and limiting accuracy to a certain granularity via binning. However, this work does not study the longevity, or the nature of hidden services and how they are used. This is mainly because reporting such statistics, might harm the privacy of individual Tor users.

The closest study to our work is PrivEx [116], which our privacy preserving counting protocol is based upon. In this work, the authors introduce two variants for secure counting. One with a shared secret key and another one that relies on multi-party computation. Our work uses the second variant. The authors implement and deploy their protocol to collect and study egress traffic from Tor, and case study the popularity of certain Internet website in different locations. Such work can be used to investigate the censorship of content in different locations, in a privacy preserving manner. PrivCount [117] is another work that relies
on PrivEx, for measuring the Tor network designed with user privacy as a primary goal. It aggregates the collected data from different Tor relays to produce differentially private outputs.

Another work [118] uses Tor circuit and website fingerprinting techniques from the middle relays to detect the hidden services popularity and usage. Furthermore, the authors use their fingerprinting technique to study the popularity of a social networking hidden service, by setting up middle relays in the Tor network. The authors demonstrate the possibility of such passive study in a privacy preserving manner from the middle relays. In another work [119], the authors introduce Histor, which relies on differential privacy to collect and analyze information about Tor statistics.

Privacy preserving techniques are also being used in machine learning and deep learning algorithms for secure aggregation of high-dimensional data. For example, [120] allows the computation of sum of large values at the honest-but-curious and active adversary server, using the data from different clients. Dolev et al. [121] introduce privacy preserving algorithms for computation and performing search, fetch, and range queries with the map-reduce framework. Related to our work is also Prio [122], a privacy-preserving system for the collection of aggregate statistics.

5.2 Tor Safety Board

Tor’s safety board [114] is established to oversee the ethics and privacy concerns of research on Tor and hidden services. It consists of a number of researchers and members of the core Tor people. Researchers interested in studying Tor and hidden services must submit a research proposal, outlining their objectives and methodologies. The proposal should justify the merits of the work, outweighing the potential risks and privacy concerns. The main goal of the Safety Board is to “minimize privacy risks while fostering a better understanding of the Tor network and its users” [114]. The safety board provides a set of guidelines for ethics and safety of the research projects.

We needed the approval from the safety board, to be able to setup a large number of HSDir relays. The bad relays group is responsible for the detection and elimination of adversarial and snooping relays from the Tor network. Controlling
a large number of relays without coordination and approval from the core Tor people results in blocking of relays.

To ensure that we are not compromising the security and privacy of Tor, we took the following procedures. We setup a website [123] detailing our project, methodology, fingerprints of our relays, and the identities of the research team. We limit the bandwidth of our relays to 250KB/s, to prevent acquiring the Guard flag. Additionally, we disable the egress traffic to the Internet from our relays, to avoid acquiring the Exit flag. The Exit nodes are specially important component of the Tor network, because they have access to the traffic (if not encrypted). They are also a target of abuse complaints, since their IP is listed as the source of connection to any Internet domain. Furthermore, if the same entity is chosen as both the Guard node and Exit node, he can perform network correlation attack and de-anonymize Tor users. Therefore, we listed all of our relays as one Effective Family, to prohibit two of our relays to be chosen in any one circuit. This ensures that our relays only serve as Middle relays and HSDirs inside the Tor network.

We have been among the first researchers to undergo Tor Research Safety Board committee review. A copy of our proposal is publicly available online [124] on the Tor Research Safety board website. The response, approval and feedback that we received is also publicly available [125]. We hope our research serves as an example for safe and ethical future works on Tor security and privacy infrastructures.

5.3 PrivEx: Privacy Preserving Counting

Our privacy preserving data collection and aggregation is based on the second variant of PrivEx [116], using distributed encryption and multi-party computation. In the following, we describe the algorithm and protocol in detail. We also provide an extension by using shuffling to avoid leaking information about a specific hidden service. To avoid confusion we use the same notation and terminology as PrivEx.

The multi-party computation relies on two sets of nodes, the Tally Key Server (TKS) nodes and the Data Collection (DC) nodes. In the context of our work each DC node is an HSDir, and each TKS is a research group, controlling a set
of the HSDirs. In our model TKSs do not necessarily need to operate or control a DC.

Each TKS generates a (private, public) key pair, which the group public key is based upon. The DCs use the group public key to encrypt the counts using a variant of ElGamal encryption with (private, public) key pair, \((a, A = g^a)\), where encryption of a message \(m \in \mathbb{Z}_q\) and a random value \(r \in \mathbb{Z}_q\) is 
\[
E_A(r; m) = (g^r, A^r \cdot h^m).
\]
In this scheme, \(g\) and \(h\) are the generators of a group \(\mathbb{G}\) of order \(q\). Note that this scheme is an additive homomorphic encryption construct, where
\[
D_a(E_A(r_1; m_1)) \cdot D_a(E_A(r_2; m_2)) = D_a(E_A(r_1 + r_2; m_1 + m_2)).
\]

The decryption of cipher tuple \((C_1, C_2)\) is, 
\[
D_a(C_1, C_2) = DL_h(C_2/C_1^a),
\]
which is based on the calculation of discrete logarithm. Given that the size of message space is small in our experiments, it can easily be calculated with brute-force method. A count value cannot be larger than the duration of the study, \(d\). Even if we control all HSDirs in the Tor network, the total count cannot be larger than \(6 \cdot d\). For further optimization and more efficient calculation of discrete logarithm, other general approaches such as baby-step giant-step [126], Pollard’s kangaroo algorithm [127], or Pollard’s rho algorithm [128] can be used.

We use a public bulletin board (PBB) for storage of all the intermediary and final results of the computations that are accessible to all DC and TKS nodes, to read from and write to. To mitigate against misbehavior and data manipulation by the PBB, all data can be either signed by the publishers, or the PBB can act as an append only database or ledger. We use a private git data repository where each DC and TKS have their own private key to publish their data.

The privacy preserving data collection and aggregation, follows a set of steps and phases that are outlined below.

**Setup:** At the setup stage each TKS picks a random value \(a\) to be used as its
private key, \( a_i \in \mathbb{Z}_q \). The public key of the corresponding private key is calculated as, \( A_i = g^{a_i} \). To mitigate against an adversarial TKS, each TKS will first publish its commitment. Only after all TKSs have published their commitments, each TKS will submit its public key. At this stage, the public keys will be verified against the commitments. If they are valid the protocol proceeds to the next step, otherwise it will be halted. The group public key \( A \), is calculated as \( A = \prod_i A_i \). At this stage, each DC also verifies the public keys of the TKSs.

**Counting:** Upon receiving the descriptor of a hidden service, each DC first populates its database with a predefined noise level by encrypting an initial \( n \), \( E_A(r; n_h) = (g^r, A^r \cdot h^{n_h}) \). A new random value \( r \) is chosen for each hidden service. \( n_h = 0 \) can be used to avoid adding noise. If the hash of the .onion address already exist in the database, the DC increments the counter by calculating \( E_A(r; 1) = (g^r, A^r \cdot h) \) and by homomorphically adding the value to the current encrypted count in the database. We use an epoch of 24 hours to consider a new hidden service. After a hidden service is observed by a DC (HSDir) \( c \) times, the value for the counter will be \( (g^r, A^r \cdot h^{c_h + n_h}) \).

**Aggregation:** At end of every round, the DCs publish their commitment to the PBB. After all DCs have committed, they publish their encrypted counts to the PBB, for each onion address \((H(\text{onion}), (g^{r_h}, A^r \cdot h^{c_h + n_h}))\). The TKSs verify the counts against the commitments and if successful continue to the next step. At this stage, each TKS aggregates the counts by componentwise multiplication of the encrypted counts, where \( (\alpha_{i,h}, \beta_{i,h}) = (g_i^{r_h}, A_i^r \cdot h^{c_h + n_h}) \), for each DC, \( i \), by computing \( \alpha_h = \prod_i \alpha_{i,h} \). Now each TKS \( j \) computes its share of decryption, \( \alpha_h^{(j)} = (\alpha_h)^{a_j} \), where \( a_j \) is the private key of each TKS. The decryption shares are posted to the PBB.

**Decryption:** After each TKS publishes its contribution of decryption, each entity can decrypt the counters by calculating the discrete logarithm. Given that each count cannot be larger than the duration of the study, one can use the brute-force method to calculate the discrete logarithm, or any of the faster general algorithms mentioned above.

Note that we use a \( k \) out of \( n \) encryption approach, where \( k = n \). If any of the parties aborts or does not participate, the encrypted data cannot be decrypted. We choose this approach for our security requirements, specially given our jurisdictions.
To comply with Tor’s standard ethics we do not store the raw hidden services at any point. Each DC calculates the hash of an onion service, $H(\text{onion})$. This mitigates against leaking the hidden services address to any adversary without prior knowledge of the onion address. However, it does not protect against an adversary with prior knowledge of a hidden service. One approach would be to use a keyed hash at the DCs, where the keys are calculated using any distributed multiparty key generation approach. The TKSs do not need to have the knowledge of the key.

We protect against a curious adversary, by providing an extension to PrivEx in our protocol. In this variant, we do not have access to the individual counts for a hidden services. Therefore, the results from each TKS cannot be compared to another TKS for an individual hidden service. However, all honest parties would generate the same histogram at the end of the protocol. The shuffling stage takes place after the aggregation phase and before the decryption stage.

**Shuffling:** We assume the number of hidden services that we observe during the experiment is $N$. We calculate the length of a fixed size array, based on $N$, to serve as our hash table. The number of collisions in the hash tables depends on the size of the hash table and the number of hidden services, and can be tuned depending on the requirements. In this setting, each TKS after the aggregation step, would re-encrypt and shuffle the data using verifiable secret shuffling [129, 130] and publish a zero-knowledge proof of equivalence of the values. At the decryption stage all parties generate the same histogram.

5.4 **Extrapolation: Inferring Longevity from the HSDir Counts**

In this section, we review our approach to infer the longevity of hidden services by extrapolating from the counts at the HSDirs. We use simulations by mimicking the process of the hidden service placing descriptors in the Tor network, according to the Tor’s hidden service specification, to estimate the distribution of expected counts at HSDirs.
5.4.1 Expected Counts at HSDirs

If each hidden service was only placed on a single random HSDir, the probability for each HSDir to host a hidden service is \( p_0 = \frac{1}{N_{\text{hsdirs}}} \), where \( N_{\text{hsdirs}} \) is the number of HSDirs. The number of HSDirs is publicly available and can be found on the Tor’s official metrics website. Since there are two descriptors, derived independently, this is equivalent to doubling the number of potential hosting HSDirs. As each descriptor is placed on a set of three adjacent HSDirs, the probability of a descriptor being hosted on an HSDir is approximated by \( p \approx \frac{6}{N_{\text{hsdirs}}} \). At the end of each 24 hour epoch, we would count each hidden service twice, because not all the hidden services change HSDirs at the same time. If a hidden service is online for \( d \) days during our experiments, the average number of counts for a hidden service per HSDir relay, is as following.

\[
C_h^d = \frac{6 \times d \times 2}{N_{\text{hsdirs}}}
\]

In the above equation \( C_h^d \) is the average expected count of hidden services \( h \) that has been online for \( d \) days during the experiment. Given that we control \( N_{\text{control}} \) of the total HSDirs, the average total count for each hidden service \( (C_h) \), after \( d \) days, from our HSDirs is outlined below.

\[
C_h^d = \frac{6 \times d \times 2 \times N_{\text{control}}}{N_{\text{hsdirs}}}
\]

This formula gives us the average expected count values for each hidden service from our HSDirs. To calculate the distribution of the counts, we use simulations to find the expected observation counts as a function of the longevity of a hidden service for the duration of the our study. Figure 5.1 shows the expected values for lifespan of hidden services. We calculate the mean, one standard deviation, two standard deviations and 95% of the values from the mean value. The distribution of expected counts follows the bell curve distribution. As we can see, 95% of the expected values are within two standard deviations of the mean expected value.
Figure 5.1: Expected counts at HSDirs as a function of longevity. As we can see, 95% of the expected values are within two standard deviation of the mean expected value.

Figure 5.2 depicts the histogram of expected values for different lifespans of 10 days, 30 days, 50 days, and 90 days. The results are based on 10000 runs of simulation. We simulate the placement of the descriptors on 6 relays according to Tor specifications for \(d\) days. At the end of each run of the experiment, simulating a hidden service with longevity \(d\), we count the number of times that the descriptor is uploaded to the HSDirs under our control. The Y-axis, represents the frequency percentage for each expected count. As it is evident, the longer the lifespan of a hidden service, the closer its distribution of the counts to the bell curve distribution. As can be seen in Figure 5.2a (10 days) compared to Figure 5.2d (90 days). Moreover, the hidden services with shorter lifespan have a lower standard deviation and range of expected counts. This explains the large distribution of values around the expected counts of 1, 2, and 3.

### 5.4.2 Weighted Frequency Count

One approach to infer the longevity of hidden services, is to use the mean expected count at HSDirs. However, this approach ignores all other possible expected
Figure 5.2: Histograms of expected values for different lifespans, 10 days, 30 days, 50 days, and 90 days. The results are based on 10000 runs of simulation. The Y-axis, represents the frequency percentage for each expected count.

counts, and certain values with higher probability of observation, dominate and skew the results.

As shown in Figure 5.1, 95% of the counts are within the two standard deviations of the mean. By only using the mean value, majority of the expected counts are not accounted. For a better representative extrapolation of the longevity, we use a weighted sum of the distribution of different longevities. For example, imagine for each longevity \(d_i\), we expect to have the count \(x_{i,j}\) with probability \(p_{i,j}\), \((x_{i,j}, p_{i,j})\). Then for each \(x_{i,j}\), we count \(p_{i,j}\) hidden services with lifespan of \(d_i\) days. In this way, we compute a weighted inverse of the observed count.
<table>
<thead>
<tr>
<th>Country</th>
<th>City</th>
<th>Cloud Provider</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>Boston</td>
<td>NEU</td>
<td>20</td>
</tr>
<tr>
<td>USA</td>
<td>Chicago</td>
<td>Vultr</td>
<td>2</td>
</tr>
<tr>
<td>USA</td>
<td>Dallas</td>
<td>Vultr</td>
<td>2</td>
</tr>
<tr>
<td>USA</td>
<td>Atlanta</td>
<td>Vultr</td>
<td>2</td>
</tr>
<tr>
<td>USA</td>
<td>Miami</td>
<td>Vultr</td>
<td>2</td>
</tr>
<tr>
<td>Germany</td>
<td>Frankfurt</td>
<td>Vultr</td>
<td>6</td>
</tr>
<tr>
<td>UK</td>
<td>London</td>
<td>Vultr</td>
<td>6</td>
</tr>
<tr>
<td>France</td>
<td>Paris</td>
<td>Online SAS</td>
<td>32</td>
</tr>
<tr>
<td>Netherlands</td>
<td>Amsterdam</td>
<td>Online SAS</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 5.1: The distribution of our relays per country, city, and cloud provider. We chose diverse location for the relays, both for security reasons and also to help the Tor network by contributing relays and bandwidth.

5.5 Implementation and Deployment

In the following section, we describe our implementation and deployment of the privacy preserving data collection and analytics. We took the procedures necessary to ensure the security and privacy of the data collected.

We deployed 80 relays that were managed by three different teams, two from the USA and another one from Europe. We divided our relays between the teams at different geolocations for security and privacy reasons. 40 relays (20 VM) were managed by Northeastern University, 20 relays (10 VM) managed by SUNY Buffalo/University of Central Florida, and 20 relays (10 VM) by Airbus. We hosted two relays per VM instance. We used a combination of private and public cloud to host our relays. 20 relays (10 VM) were hosted at Northeastern University, 20 relays (10 VM) on Vultr cloud provider, and 40 relays (20 VM) on Online SAS. We chose these two cloud providers because of low cost, and that both are in the top 5 cloud providers used by the Tor relays. These providers have data centers in multiple geolocations, allowing us to distribute our relays, both for security reasons and also to help the Tor network. The 20 Northeastern University relays, were hosted in Boston, USA; 32 Online SAS relays in Paris, France; 8 Online SAS relays in Amsterdam, Netherlands; 6 Vultr relays in Frankfurt, Germany; 6 Vultr relays in London, UK; 2 Vultr relays in Miami, USA; 2 Vultr relays in Atlanta, USA; 2 Vultr relays in Chicago, USA; and 2 Vultr relays in Dallas, USA. Table 5.1 summarizes the distribution of our relays per country, city, and cloud.
provider. Please note that the count is the number of the relays, not the number of VM instances. There are two relays per VM instance, since only two relays can run on a single IP address, per Tor policies.

We made modifications to the Tor software to log the hash of the hidden services and the time they were uploaded to the HSDirs into an in-memory database. For privacy and security reasons, we used the Redis in-memory database, with the persistence storage disabled. As a result, as soon as the machine is restarted, all unencrypted data that resides in the memory will be deleted. We implemented the privacy preserving counting and aggregation in Python. At the end of each 24 hour epoch, the Python program reads the data from the Redis database, writes the encrypted data to disk, and flushes the databases. At the end of each cycle, we publish the data to the PBB. We used a private git server as our PBB, to collect data from all DCs. When the data collection phase is over, all TKS aggregate the counts, calculate their contributions and publish it to the PBB as well.

Note that the encrypted data on the DCs, cannot be decrypted until all parties participate. The reason behind using a $k$ out of $n$ (where $k = n$) encryption technique is to ensure the security of the collected data. Even if parties are coerced in disclosing their private keys, as long as one party deletes his private key, the encrypted data cannot be decrypted. Moreover, both our DCs and the TKSs reside in different geolocations with different jurisdictions, which makes the coercion of the private keys less practical.

Figure 5.3 depicts the geolocation distribution of our HSDir relays. The green marker is CCIS NEU, the red markers are Online SAS, and the blue markers are Vultr servers. As evident, we distributed our servers to different geolocations for security and privacy reasons. Furthermore, we helped the Tor network by contributing relays and bandwidth at different data servers.

## 5.6 Results

In this section, we discuss the results and findings of our study. As mentioned previously, we deployed 80 HSDirs over a range of public and private cloud providers. Our experiments lasted for 90 days, from September 15, 2017 to December 15, 2017. During this period, we collected the longevity counts for 152284 hidden
Figure 5.3: The geolocation map of our relays. The green marker is CCIS NEU, the red markers are Online SAS, and the blue markers are Vultr servers. We distributed our servers to different geolocations for security and privacy reasons.

services from our relays. We also collected the popularity measure for hidden services from our relays.

Figure 5.4 plots the distribution of the extrapolated longevity and the raw longevity counts from the HSDirs. As we can see around 40% of the collected hidden services have count of 1. However, in the extrapolated longevity graph, the majority of the hidden services have an estimated longevity of 2 days. This is because if a hidden service has a count of 1, it can be, with a high probability, a hidden service with longevity of 1, 2, 3, or higher. However, if a hidden service is counted 2 times, it can only have a minimum longevity of 2 days. This imbalance between the estimated longevity and the observed counts for values less than 10, explains the different distribution shape between the raw longevity counts and the estimated lifespan of hidden services.

Figure 5.5 depicts the CDF of the observed counts and the estimated lifespan. More than 50% of hidden services are observed 2 times or less on the HSDirs. At the same time, more than 50% of the hidden services, have an estimated lifespan of less than 11 days, where 40% of them are online for only less than a week. Our results show that a large majority of hidden services, have short lifespan, less than a few weeks. Our findings confirms the results of previous work [3], regarding the number of hidden services with short life span. However, the previous work did not consider any safeguards, and their results were based on the probing of the collected onion address from their HSDirs, which is against Tor user agreement. In contrast, our work does not store any information about the onion services.
Figure 5.4: Distribution of the estimated longevity and raw longevity counts from the HSDirs.

Figure 5.5: CDF of distribution of the estimated longevity and raw longevity counts from the HSDirs.
Furthermore, we do not perform any probing of the hidden services. Our results rely on the extrapolation of longevity based on the counts from a small number of hidden service (2% of the hidden services).

Without a holistic view of all the hidden services (i.e., controlling all HSDirs), it is difficult and non-trivial to draw conclusions on the churn of the hidden services. However, by using the hidden services statistics that the Tor project collects from all relays and the lifespan of hidden services, it can be inferred that the hidden services have a high churn. A possible explanation for such short-lived hidden services with high churn can be the dynamics and behavior of the applications that use hidden services. For example, OnionShare, a secure and private file sharing that uses Tor hidden services, creates a new random hidden service address for each file transfer. As soon as the file is downloaded by the recipients, the hidden service is discarded.

To give us some insight into the possible use cases of the hidden service, and draw better conclusions on the nature and intent of them, we measure the popularity of each service and compare it to its longevity. Figure 5.6 depicts the popularity of hidden service as a function of their lifespan. Majority of the hidden services have a low popularity, regardless of their lifespan. This behav-
ior is expected, since the overall number of Tor clients and users is not large. Additionally, many services do not have a presence in the dark web.

As we can see, hidden services can be grouped into five categories based on their lifespan and popularity: (1) hidden services with low popularity and visit, regardless of their lifespan (dark blue); (2) hidden services with low longevity, and very high popularity (red); (3) hidden services with short lifespan, but high popularity (magenta); (4) hidden service with average longevity, and average popularity (cyan); (5) hidden services with high lifespan and high popularity (orange). Groups 2 and 3 (short lifespan, but high/very high popularity), can be hidden services that need more exploring in terms of their behavior and activity.

The short-lived, but very popular hidden services, can be domains where they host a malicious content that users try to connect to before they disappear. This content can be either websites, forums, or OnionShare addresses for file sharing. Currently, there are not many legitimate hidden service candidates, that could justify such visit patterns. The hidden services with long longevity, can expect a high popularity and visit rate considering the time needed to gain a large user base. Examples of such domains are NYTimes, Facebook, and SecureDrop. Note that a large majority of hidden services are not popular, regardless of their lifespan. This can be an indication that hidden services are not yet fully adapted by users and service providers.

5.7 Protection Against Dishonest Adversary

In this section, we describe and discuss different approaches and solutions to detect and mitigate against a dishonest DC and TKS, who report fake counts. A dishonest DC and TKS, can skew the histograms and final counts by reporting fake values.

Approach 1: In this setting, we assume a group of entities that each one runs $x$ relays. Our objective is to detect cheating by the other groups, using a pairwise comparison. In this model, each entity can set their own trust threshold ($\delta$) to include the reported counts from others. For example, if entity $A$ owns $x_A$, and entity $B$ owns $x_B$, with thresholds of $\delta_A$ and $\delta_B$ respectively, they can perform a pairwise comparison of the counts of their HSDirs. For simplicity, we consider cases where $x_A = x_B$. Otherwise, one should extrapolate the counts if the number
Figure 5.7: Graphs of count matches between two entities, where each controls $x$ relays. The X-Axis is the number of relays each entity controls, and the Y-axis, is the probability that both entities have a matching final count, with absolute difference of 0, 1, 2, and 3.

of controlled HSDirs is different. Figure 5.7 shows the probability of two sets of HSDirs, with $x$ HSDirs per group, which have matching counts for a hidden service with longevity of 10, 30, 90, and 120 days, and $\delta = 0, 1, 2$.

The smaller the number of HSDirs owned by each party, the higher the probability that their counts would match. Because by increasing the number of controlled HSDirs per party, we are also increasing the probability that one party owns the majority of the responsible HSDirs for a hidden service. Please note this is the case for random selection and assignment of the relays. If the three entities can control relays alternatively, and the network formation stays static, then the counts at different entities will be a perfect match. However, this is not a realistic model for Tor relays operations and dynamics. Relays are randomly
created and controlled.

As we can see in Figure 5.8, for a hidden service with longevity of 90 days, as we increase the number of controlled HSDirs to 1000 (one third of all the HSDirs), the probability of a match between two HSDirs drops to less than 5%. Imagine each party controls 80 HSDirs ($N_{control}$) out of the 3000 ($N_{hdirs}$) total HSDirs. Then, the average expected count of each party, for a hidden service with longevity of 90 days ($d$) is 14.4, according to the formula $c = \frac{6 \times d \times N_{control}}{N_{hdirs}}$. However, if we increase the number of control HSDirs to $N_{control} = 1000$, the average counts would be 180. Therefore, the possibility of a match between the two counts drops significantly; even if we consider the absolute difference of up to 3 between the counts.

Figure 5.9 depicts the histogram of the difference of the aggregate counts between every two entities that controls HSDirs. As we can see, around 80% of the count differences are less than or equal to 5. Imagine the threshold of the difference to be $\delta$, and the probability of a count within the bounds to be $p_0$. Then, the probability that $i$ out of the $n$ counts are outside the threshold, would be, $(1 - p_0)^i$. A TKS can set the parameters to establish a level of confidence on the correctness of the counts.

Note that this setting can reveal information about individual hidden services, if no other protections are used. However, if we use shuffling, then each hidden service bucket does not reveal information about an individual hidden service. This is because a set of possible hidden services map to the same index.

**Approach 2:** In this approach, we create a set of private hidden services without announcing them to the other operators. By running these hidden services, each party knows exactly which HSDirs have hosted their descriptors at any time. To verify if other operators are honest, he can ask for the decryption of the count of his hidden service after proving that he owns the private key of the specified hidden service. Each party $p_i$, owns $k$ RSA (private, public) key pairs $(d, e)$, and runs $k$ private hidden services $H_k$. Before accounting any DC’s contribution, each party first proves the ownership of $H_k$, using his private key. After the ownership of a hidden service is verified, each $DC_i$ submits his count for $H_k$. The owner of the hidden service knows the exact count of his hidden service for each $DC_i$. If the reported number from the DC is equal to the value, then the DC counts the $DC_i$ contribution. Given that the DCs have committed to their counts before publishing them on the PBB, they cannot selectively cheat. Meaning, they
Figure 5.8: Count match between two parties, controlling up to 1000 (1/3 of all relays), for a hidden service with longevity of 90 days. As we can see, the probability of a match decreases as we increase the number of control HSDirs.

Figure 5.9: Histogram of count differences between every two sets of HSDir entities.
cannot report correct values for the honeypot hidden services, and wrong values for all other hidden services.

Imagine that a DC is honest, but the counts do not match with probability $p_0$. After querying $i$ honeypots, the chances of non-matching counts is less than $(1 - p_0)^i$. By setting the parameters for different adversarial models and context, a TKS can establish a level of confidence on the correctness of the counts.

Note that if the honeypots return an empty page, or have any other deterministic characteristics, an adversary can simply probe the hosted hidden services and verify if they are honeypots. Additionally, the adversary can report false values for the known hidden services that are not hosted by the verifying parties (e.g., Facebook). One possible solution is to ignore the counts for the honeypots that are visited, since they are private. A visit indicates that possibly an HSDirs is probing them. In this approach, we can have a set of honeypots that are not visited by the HSDirs and we can use them to detect the dishonest DCs.

Approach 3: Considering that an adversary can detect the honeypots, another approach is to verify the counts of a set of all hidden services. This is a variant of the second approach. We sample a random set of all online hidden services, and query each entity for their corresponding counts. In this method, the adversary is not able to report fake values on the known hidden services that do not belong to the verifiers. Each verifier checks the longevity of any hidden service that is uploaded to his HSDir, by performing a non-intrusive probing. Such probing is carried out by establishing a connection to the hidden service, without making an HTTP request. As we saw in Figures 5.7 and 5.8, an honest HSDirs reports counts $c$ with the threshold $\delta$, with probability $p_0$. In the figures and simulations the HSDirs formation is static. Imagine that in a dynamic HSDir circle formation, the probability that an honest HSDir reports a count outside the threshold is $p_1$. Therefore, after $i$ iterations of sampling hidden services longevity, each TKS can establish a level of confidence on the correctness of the counts. In this model, each TKS entity can set their own threshold $\delta$ and only consider the counts that meet their security requirements.
5.8 Alternative Approaches to Secure and Private Computation

Another direction of research for secure and privacy preserving computation is based on the assumption of availability of secure hardware and trusted computing bases such as Intel SGX. For example, [131] uses isolated execution environments (IEE) by Intel SGX, to provide secure multiparty computation (MPC) protocols. In this protocol, SGX plays the role of a trusted third party, to attest and bootstrap secure communications between participants. In this scheme, the communications and computations overhead depends on the size of input. Another example is [132], which aims to provide privacy preserving data analytics on a shared platform where data security and trustworthiness of computations are ensured by the hardware such as Intel SGX. Researchers have proposed techniques that carefully design data-independent access paths. Such techniques help protect against side channel attacks such as cache access, CPU usage, and other timing attacks that leak private information. At the same time, many of these techniques introduce significant overhead, which makes them prohibitive in large-scale real-time data analytics. The authors introduce new techniques to provide a trade-off between privacy protections and speed, by adding noise to traces of memory access.

Another domain where Intel SGX can be used is for genomic and DNA sequencing. Currently, the increase of DNA sampling rates has resulted in the production of significant amount of data. This increase has led to the use of public cloud infrastructure for alignment of raw genomic information. However, such work needs to be carried out in a privacy preserving manner, both for ethical and legal issues. This is mainly because such studies produce personalized medicine and diagnosis that should stay private. Works such as [133], use Intel SGX for alignment of raw genomic information in privacy preserving manner, to speed up the process. The limited secure enclave memory introduces new challenges in this context, given that the data structures far exceed secure enclave memory.

As mentioned, secure hardware components such as Intel SGX, are susceptible to side channel and timing attacks [134, 135]. Furthermore, in the light of vulnerability discoveries such as Spectre [136] and Meltdown [137], the trusted computing based security assumptions that SGX security relies upon are compromised. As a result, solutions that rely on cryptographic constructs and security
models are more desirables, compared to protocols that rely on isolated and trusted computing bases such as Intel SGX.

5.9 Conclusion

Tor and hidden services have attracted a much larger user base in the past few years. Investigating the hidden services dynamics and pro-actively protecting them against abuse and subversion is crucial in their success. Given the strict privacy requirement and security assumption of Tor, it is important to consider the implication of data collection and analysis in such infrastructure.

In this chapter, we proposed new protocols and approaches to study the longevity of hidden service in a privacy preserving manner. We deployed a small number of HSDirs to infer the lifespan of hidden services. As we saw, a large majority of hidden services have short lifespan and high churn. We also looked at the popularity of hidden services as a function of their lifespan and provided a categorization of their dynamics. Furthermore, we discussed how the hidden service lifespan can be used as an indicator for malicious or benign activity. Such approaches allow the protection of Tor against abuse, while respecting the privacy of the legitimate users. Future work on large scale data collection and analysis of any privacy infrastructure, should consider these ethical and privacy concerns.
Chapter 6

Future Work and Outlook

As we discussed in this thesis, privacy infrastructures can be subverted and abused. Our work on OnionBots, discussed how privacy infrastructures can be abused for a next generation resilient botnet. HOnions, were conceived as a mean to expose and detect malicious actors who setup snooping HSDirs. With privacy preserving longevity study of the hidden services, we depicted how one can develop proactive methods to study hidden services and possibly detect malicious actors, without compromising the security and safety of the benign and legitimate users.

Today, with the increased penetration of Internet and technology, security and privacy are of paramount importance. Because of the increased rate of data generation and collection, more people are concerned about their privacy. Furthermore, the numerous revelations about governments, companies and employers collecting data on their citizens, customers and employees, to build a behavioral profile on them has increased such concerns. According to the Tor Statistics [83] the number of Tor users has increased by twofold in the past year alone (Figure 6.1). More than 40% of the users are from Germany (30.91%) and United States (13.31%), two advanced countries with high Internet and connectivity penetration index. Such need for privacy has motivated previous work to explore a new outlook for Internet connectivity, where instead of running Tor as an overlay over IP network, to integrate privacy and censorship resilience into the Internet. However, there are many challenges such as robustness to fail over, efficiency compared to the current Internet technology, and ISP economic models. [138].

In the following, we briefly look at the few directions of future work in the
Figure 6.1: Number of Tor users based on the statistics provided by Tor Metrics.

context of privacy tools and infrastructures.

## 6.1 Subversion and Abuse of Privacy Tools and Infrastructure

The adversaries and malicious actors can subvert and abuse any technology. Specially, privacy tools and infrastructures are an attractive target, because of the services and safety guarantees they provide. Recently, we witnessed numerous accounts of such abuse, such as the rise of ransomwares, which is simply a trivial abuse of cryptographic primitives. Furthermore, dark web market places, such as SilkRoad, are a combination of abuse of Bitcoin and Tor hidden services. Bitcoin is a cryptocurrency that allows payments and transfer of value from one entity to another, without strict association and attachment to real world identities.

Moreover, infrastructures such as WiFi networks and access points (AP) and Internet of Things (IoT) devices can be abused for spread of an undetectable airborne malware [61], form a botnet for distrusted denial of service attack [139], or injection of malicious code to mine cryptocurrencies [140].
The aforementioned cases are examples of actors who abuse privacy tools for personal gains. Our work on OnionBots describe possible solutions to address such problems.

Another cluster of malicious actors, referred to as “malicious privacy enablers”, exploit the users of privacy infrastructures. For example, malicious exit relays who perform man-in-the-middle (MITM) attack on users’ traffic [2], or malicious HSDirs who snoop on users’ hidden services. Our work on HOnions [104], proposed a new framework for detection and identification of such malicious actors. More work is needed to investigate the dynamics, motivation and behavior of adversaries.

6.2 Ethical and Safe Study of Privacy Infrastructures

As evident from previous studies [110, 3], where adequate safeguards and privacy protection for users were not taken into account, it is crucial that more researchers would uphold ethical standards and intrusive studies [84] will not be carried out in the future.

Our privacy preserving longevity study of hidden services is among the first studies approved by the Tor Research Safety Board [114]. Our work has already been used as an example for the procedure to setup such studies, where the results further demonstrate the feasibility of accurate and meaningful data collection and analysis. We hope more future work on studying the privacy infrastructures such as Tor, consider the ethical issues and get approvals from a committee such as Tor Safety Board to ensure they are not compromising users’ security and privacy.

6.3 Privacy in Large Scale Data Collection and Analysis

The significant increase in the rate of data generation and collection, has caused the rise and success of cloud computing. In the cloud computing paradigm, the storage and processing of data is done remotely over untrusted third party
servers [141]. The need for privacy has ignited research on privacy preserving computing over untrusted platforms. A fully homomorphic encryption [142] is the holy grail of such endeavor. However, current technology is inefficient and slow for any real world applications. The lack of such theoretical framework, has resulted in the development of secure hardware modules such as SGX [143] that carry out trusted code execution on untrusted platform [107]. Recent works have investigated the use of SGX for private and secure computation and analytics [132, 131], and DNA genome matching [133]. However, SGX has shortcomings such as side channel attacks [134, 135]. In the light of vulnerabilities such as Spectre [136] and Meltdown [137], the notion of their trusted computing base (TCB) is challenged.

Another direction of research, investigates techniques that do not rely on secure hardware such as SGX, while providing many security and privacy guarantees. For example in this thesis, we used an additive homomorphic encryption, and multiparty computation to study the longevity of hidden services. Other works in this framework, include [120] and [144, 145], where multiparty computation is combined with homomorphic encryption schemes.

Recent events such as the Strava revelations [146] are a reminder that the collection and analysis of large data sets can be compromising, even in an environment where privacy is not the main functionality. Collection and study of data from a privacy infrastructure such as Tor is even more sensitive, and requires thorough investigation. The findings of our research can be used for the study of other sensitive, personal, and private information such as medical and health records.

6.4 Next Generation Hidden Services

In this thesis, we explored few aspects of the security and privacy issues of the current Tor design. The next generation Tor hidden services should, in principle, be capable to address certain shortcomings. For example, under the new design the raw hidden service address, is not accessible to the HSDirs. The addresses are encrypted with a key that is a function of the hidden service address. Therefore, if an HSDir does not have a prior knowledge of the raw hidden service address, it will not be able to decrypt the descriptors that are uploaded to it. This improvement stops the snooping of the hidden services. However, the new design
is still susceptible to be abused by botnets, and facilitation of access to malicious content. One of the problems with the new hidden service is the much longer domain names (56 characters instead of 16 characters), which introduces a new attack vector for the domain names. Future work and research are needed to protect such privacy infrastructures against abuse. Furthermore, like any other new design and implementation, the next generation design of hidden services needs to be thoroughly tested and investigated to find, and ultimately solve the potential vulnerabilities.
Bibliography


105


[99] Tor, “Five years as an exit node operator,” https://blog.torproject.org/blog/five-years-exit-node-operator.


