UNIVERSITÀ DEGLI STUDI DI MILANO
Facoltà di Scienze Matematiche, Fisiche e Naturali
Dipartimento di Informatica

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE
XXVIII CICLO

Active Techniques for Revealing and Analyzing the Security of Hidden Servers

Ph.D. Candidate
Srdjan Matic

Adviser
Prof. D. Bruschi

Co-adviser
Dr. J. Caballero

Ph.D. Coordinator
Prof. P. Boldi

Academical Year 2015/2016

Abstract of the dissertation

In the last years we have witnessed a boom in the use of techniques and tools that provide anonymity. Such techniques and tools are used by clients that want their communication to stay anonymous or to access censored content, as well as by administrators to hide the location of their servers. All those activities can be easily performed with the support of an anonymity network. An important component of an anonymity network is the hidden server, a machine whose IP address is kept secret. Such hidden servers are the target of research in this thesis. More specifically, we focus on different types of hidden servers used in the Tor anonymity network. Tor hidden services (HSes) are anonymous services hosted in the Tor Network. The HS itself is a hidden server because users that connect to it are not aware of its IP address, and thus its location. Another equally important kind of hidden servers are Tor bridges. Bridges are entry nodes of the Tor Network, whose IP address is not publicly disclosed to avoid blocking traffic towards them. Bridges are meant to be used by clients that connect from countries where governments perform selective filtering over the contents that users can access, and for this reason governments try to block connections to those nodes. In this thesis we develop novel approaches and we implement them into techniques to analyze the security and reveal the location of hidden servers. This thesis comprises two parts, one dealing with HSes and the other one with bridges.

In the first part of the thesis, we develop a novel active approach for recovering the IP address of hidden servers that are used for hosting HSes. To this end, we design, implement, and evaluate a tool called CARONTE that explores the content and configuration of a hidden service to automatically identify location leaks. Later those leaks are leveraged for trying to unveil the IP address of the hidden service. Our approach differs from previous ones, because CARONTE does not rely on flaws in the Tor protocol and assumes an open-world model, i.e., it does not require a list of candidate servers known in advance. A final validation

step guarantees that all the candidates that are false positives (i.e., they are not hosting the hidden service) are discarded. We demonstrate CARONTE by running it on real HSes and successfully deanonymizing over 100 of them.

In the second part of the thesis we perform the first systematic study of the Tor bridge infrastructure. Our study covers both the public bridge infrastructure available to all Tor users, and the previously unreported private bridge infrastructure, comprising private nodes for the exclusive use of those who know about their existence. Our analysis of the public infrastructure is twofold. First, we examine the security implications of the public data accessible from the CollecTor service. This service collects and publishes detailed information and statistics about core elements of the Tor Network. Despite the fact that CollecTor anonymizes sensitive data (e.g., IP or emails of bridge owners) prior to its publication, we identify several pieces of information that may be detrimental for the security of public bridges. Then, we measure security relevant properties of public bridges, including their lifetime and how often they change IP and port. Our results show how the public bridge ecosystem with clients is stable and those bridges rarely change their IP address. This has consequences for the current blocking policies that governments are using to restrict access to the anonymity network, because more aggressive strategies could be adopted. We also show how the presence of multiple transport protocols could harm bridge anonymity (since the adversary becomes able to identify the bridge through the weakest protocol). To study the private bridge infrastructure, we use an approach to discover 694 private bridges on the Internet and a novel technique, that leverages additional services running on bridges, to track bridges across IP changes. During this process, we identify the existence of infrastructures that use private proxies to forward traffic to backend bridges or relays. Finally, we discuss the security implications of our findings.

Acknowledgments

Мојим драгима,
они који су још увек ту и онима којих нема више.

Contents

1 Introduction

In modern society computers have become an essential part of our daily lives and a plethora of tasks would be inconceivable without their support. Daily communications with friends and relatives, shopping, money transactions, and even medical diagnosis is now performed while comfortably sitting at our own desk. This *digitalization* was achieved first by compacting our entire lives, in form of pictures, musical preferences, notes about our political and sexual orientation, and then uploading this information onto computers.

Because of this trend, it is not surprising that computers gradually started converting into our digital alter egos, virtual impersonations that exchange our personal information with other digital peers. What once upon a time used to be face to face talks with friends, are now conversations taking place over chats and emails. This situation in itself would not be so problematic, only if we had full control over the information that we share, but unfortunately this happens very rarely. Often, we do not have the power to choose where our data will be stored or what route it should take when it is sent over the network. This impossibility causes that, when the information is not encrypted, anyone having access to the infrastructures where our data is stored or through which is flowing, could easily examine and spy on it. And as a direct consequence of this, the amount of prying eyes has grown up rampantly. For example, in the past an indiscreet confession about an affair or a pregnancy shared only with a small bunch of close friends, might have ended up as a rumor at the ears of the bartender in the restaurant. Nowadays, the situation is completely different. This same confession, turned into a digital message, and written while sitting alone in your own office, might be intercepted and seen directly by your employer or a prying colleague.

Already a few decades ago, several start-ups identified great business opportunities in this continuous shift towards the digitalization of our lives. Some of today's biggest companies such as Google or Facebook, long time ago foresaw the potential of massive data collection and people's atavistic need to communicate. One of the main reasons of their worldwide success is precisely the intuition to

grant users a place where to store, easily access, and share with others their data, free from charges. These companies make money from ads, and the more data they have on users the better they can target advertising. On the other side, the considerable amount of collected data is also used to improve the various services they provide. Unfortunately massive data collection is not a process that concerned only companies. Several states and oppressive regimens decided to ride the wave of fear generated by the numerous terrorist attacks that characterized the last decade, and they started deploying a similar mechanism, for gathering information about their own citizens. Wielding a one-sided consensus, upon which citizens never agreed, agencies either harvest users' (meta-)data, or they tamper with the contents users want to access. In contrast with commercial services, where users *purposefully* upload and share their data, agencies tap information about users, e.g., by passively inspecting the flows in networks to which they are granted access. In case the information that is being transmitted is not encrypted, agencies will have access to the raw files that were sent. When content is encrypted, still it is possible to squeeze some useful meta-information (e.g. "who is sending data to whom?" or "how often the two entities communicate?"). Or alternatively, if a warrant is available, agencies can simply request companies to provide them with all the information they have about particular users.

In this scenario, concerns of many users, about a world that every day is becoming more and more similar to the dystopic society depicted by Orwell in his famous novel "1984", started becoming a popular opinion, especially after Snowden's revelations about NSA secret surveillance program [48]. Longing to protect their privacy and freedom of speech, many users started employing privacy enhancing technologies (*PET*s). The goal of these tools is to provide a set of utilities that will allow online users to protect the privacy of their personally identifiable information. Nevertheless, anonymous communication is not a research field discovered only recently, and the groundwork for the development of those tools dates back to 1981, when David Chaum proposed an anonymous email system [17]. From the initial design, tools for anonymous communication have significantly grown in popularity, especially in the field of *anonymity networks*. Anonymity networks, allow users to access Web and other services while hindering attempts to trace their identities online. This can be achieved by using *overlay networks* and peer-to-peer communication among nodes. Overlay networks are built on the top of the Internet and only can be accessed using a particular software. Examples of popular overlay network used for anonymous communication include Tor, Freenet, and I2P.

A core component of anonymity networks is a *hidden server*, i.e. a machine whose IP address is kept secret. Hidden servers are the ploy that is leveraged both to safeguard the identities of service owners and of the clients that interact with applications, but also for allowing users to access the anonymity network. The first step needed for making impossible to link a service to its owner is to keep its location unknown. This can be easily achieved by hosting the application on a

hidden server. While the main goal of an anonymity network is to conceal users' location and usage from attackers that conduct network surveillance, it is also important to provide the same level of protection to service owners. Often, people that make available a particular service might be interested in not disclosing detailed information about themselves. One of the reasons is that some of the contents offered by services *could be* considered "of doubtful moral" or even "illicit" in some countries. But those are not concerns that affect only lone criminals attempting to abuse anonymity to share pornographic material or to run drug and guns markets. Groups fighting for transparency and human rights are both owners and users of legitimate services that are hosted and reached through the anonymity network [115].

However anonymity networks are not only used by privacy-concerned users, but often they are the only way users have to access content that otherwise would not be available. In many countries, such as China, Philippines or Iran, states perform selective filtering on the contents that users are allowed to access [14, 65, 112, 123]. Solutions for accessing forbidden content are either to access it on a different location or to hide the fact that censored material is accessed. In this second case, users enter the network using a hidden server (which will not reveal that they are joining the anonymity network) and then, from within the anonymity network, they can freely connect to the service they are looking for. This approach has become so popular that every day thousands of users access news websites, email and social networks thorough anonymity networks [39].

In this dissertation we investigate the security of hidden servers in the *Tor* anonymity network [28]. Tor is considered "the king of high-secure, low-latency anonymity networks" [102], it consists of over 7,000 publicly known nodes and supports millions of users [110]. Because of its widespread adoption, anonymity networks and in particular Tor, have become a hot topic in the last years. Documents leaked by Snowden report about NSA running special programs to perform digital surveillance on Tor users [6,101]. There are also reports about conspicuous funding from the FBI to a university to deanonymize Tor users [77].

The Tor anonymity network was launched in 2002. The goal is to safeguard users' identities and their online activities from attackers conducting network surveillance, by separating identification from routing. Thanks to this splitting, for an attacker that has control over only one hand of the communication it becomes extremely hard to infer something about users' interests and habits. This is achieved by routing users' traffic through a set of intermediate nodes, called *relays*, that are distributed worldwide and run by volunteers. The core principle of Tor was developed in 1995, under the name of the Onion Routing project, by US military service. Differently from the its origins, nowadays Tor software is now distributed under a free software license and is used by ordinary people, whistleblower, media, political activists but also miscreants. While Tor is not illegal anywhere in the world, the access to the network is blocked in some countries (e.g., China, Iran, Philippines). Because Tor works as *overlay* over the Internet,

traffic is first split and then encapsulated into cells that are forwarded among two Tor relays that have a public IP address. Once the transmission is completed, the destination machine reassembles the cells into the original stream and pushes it to the appropriate application layer. A *virtual circuit* is a sequence of three or more relays that forward traffic from a client to a destination server. In every moment each element of the circuit knows *only the source and the destination* of a cell and does not have a global visibility on the whole circuit built by the client. This guarantees that, no matter what position in the circuit the node has, it will not be able to correlate the client with the destination of its communication. Furthermore, to achieve confidentiality, at each hop of the circuit, the data is protected using multiple layers of encryption. Each time a cell reaches a new hop of the circuit, a layer of encryption is removed, and only the last hop is able to see the plaintext information that was sent.

In Tor, hidden servers play crucial roles in different contexts. The design of the protocol allow users to offer an anonymous service by deploying it and making it accessible through the Tor Network. The application, known as *Hidden Service*[1], is accessed using a DNS-like mechanism internal to the Tor Network and the IP address of the server remains unknown to the users. In Section 1.1 we provide a more detailed overview over Hidden Services and our approach to automatically recovering of their IP addresses. We start by introducing what hidden services are and how they work, then we illustrate our tool,CARONTE, that we developed and that can be used for automatic *deanonymization* of hidden services. CARONTE works in a way that is completely agnostic from the underlying protocol and thus it can be easily adapted to work with other networks offering analogous services (e.g. "eepSites" in the I2P network). CARONTE receives as input a list of hidden services and automatically extracts potential *location leaks* that can provide information about where the service is hosted. Then it uses an algorithm for validating candidate IPs that it identified, in order to discard possible false positives. Finally, we illustrate the results of what is the first measurement study of prevalence of location leaks within hidden services.

Within Tor, another common use of hidden servers is to provide access to the Tor Network itself. Several governments are aware of anonymity networks being used for circumventing censorship, and to this end their agencies perform *active probing* on machines that are suspected to belong to the Tor Network. Once a host is confirmed being a Tor node, it is added to a blacklist in order to to not allow users to connect to it and evade the censorship [126]. Because of this, it is not convenient to have a publicly available list of all the entry points to the Network. In the Tor context, those entry nodes whose IP address is unknown, are called *bridges*. Users can contact directly the Tor Project and obtain a IP address of a limited number of bridges. Tor Project replies to each request with three IP

[1]another name for these application is "Onion Service". When referring to "Hidden Service", we will be considering both services that require the application to be hidden and those that do not have this requirement (e.g., Facebook).

address that are currently running a bridge and that the client can use anyone of them to enter the Tor Network. The reason why those nodes are considered so highly sensitive and their IPs are kept secret, is because their number is limited and they should be used only by people from countries where governments put restrictions on the material that citizens can access.

In Section 1.2 we discuss more in detail the Tor bridge infrastructure and the approach that we followed to obtain the IP addresses of those entry nodes. The extensive security analysis we performed on bridges shows how it is possible to deanonymize Tor bridges at large scale using publicly available datasets. Leveraging those datasets, it is possible to identify bridges even before they are used by clients. Similarly, we also use this public information to measure the stability of the bridge infrastructure, and our results could affect the current blocking policies that governments use when they blacklist a bridge. We expose how it is possible to perform ranking of bridges through the detailed information about each node of the Tor network, that is daily published by the Tor Project. We also show how and adversary can keep track of IP address changes of bridge, by fingerprinting other services that might be running on the same machine. Finally we investigate and measure a previously undocumented feature that we identified in the Tor network and that is represented by *proxy* hosts. Proxies do not have Tor software installed and they transparently forward connections to Tor relays and bridges. Since they are not considered elements of the Tor Network, public statistics about them are not available.

1.1 CARONTE: A Tool For Deanonymizing Tor Hidden Services

In 2004, only a few years after releasing Tor, *Hidden Services* (HSes) were introduced in the Tor Network. Tor developers wanted to allow users to access a service, without knowing its effective physical location. Recipient anonymity is achieved by incorporating the service directly in the Tor Network. Applications that ran as a hidden service, are identified by a unique name, generated randomly and ending with an ".onion" suffix. To access the service, users resolve its name using a mechanism similar to a distributed DNS, which is running in Tor Network. Soon after their introduction attacks, that leveraged some design flaws of hidden services, were proposed to identify the location of a HS [10, 86]. Because of those attacks, the protocol for accessing a HS was revised and *guard* nodes were introduced [24, 130].

The extended protocol, that includes guard nodes, requires the hidden service to select a relay, called guard node, that will be used as the introduction point for all communications with the HS. When a client resolves the ".onion" name linked to a HS, she obtains the address of the guard node. Afterwards, the client contacts the guard and asks it to forward connection information to the HS.

This initial message, containing connection information, notifies the HS what was chosen as the *rendezvous point*. The rendezvous point, is a relay that acts as an intermediary among the client and the server; its only goal is to connect the anonymous virtual circuits built from each side. The guard node is used exclusively for communicating information about the rendezvous point selected by the client; all the following messages exchanged among the client and the HS do not pass through the guard, but they directly go through the rendezvous point.

While hidden service were originally developed with the purpose of enabling freedom of speech even in situations where powerful state-based adversaries might be trying to suppress it, location privacy turned out being extremely attractive for miscreants that were planning to run illicit services. Nowadays, hidden services are used among others for storing whistle-blowing documents that corporations or governments may not want to become public and for hosting political dissidents blogs. In 2015, Facebook decided to make the social network available as a hidden service; the company also obtained a certificate for its onion domain, signed by trusted Certification Authority, so that browsers would accept it [92]. As mentioned earlier, in parallel with legitimate applications, we witnessed also the blossoming of HSes that were abused for running malware command-and-control servers [49, 62, 67]. Similar shady uses of these services include portals offering pornography and black markets selling any kind of goods (drugs, guns, stolen or counterfeit products). Taking into consideration this varied range of uses, it is not surprising that hidden services quickly attracted the attention of censors and law-enforcement agencies. What arguably happened to be the most famous take-down of a hidden service, took place in July 2013, when law-enforcement identified the location of the "Silk Road" marketplace and arrested the owner and administrator of the hidden service. Users of this service were using Bitcoin payments for trading with products such as cocaine, LSD and counterfeit goods [20]. Soon after its take-down, other competitors quickly took his place. Only one year later, in November 2014, an international joint force of law-enforcement agencies successfully dismantled a network of over 400 HSes, including several drug markets [47].

1.1.1 Problem Statement

Since their introduction, numerous attacks have been proposed for trying to break the recipient anonymity offered by hidden services [10,13,15,58,80,81,86]. Hidden services protect only the location, i.e. the IP address, of the server hosting the service. The protocol does not tamper with the data that is transmitted (except opportunistically adding and removing encryption layers, in order to protect the data transmitted among two nodes). For this reason, the configuration of the service and eventual sanitization of its contents is left to the server owner [114]. If references to the *identity* of the owner or information about the *location* of the service itself are available in the content of the hidden service, an attacker might

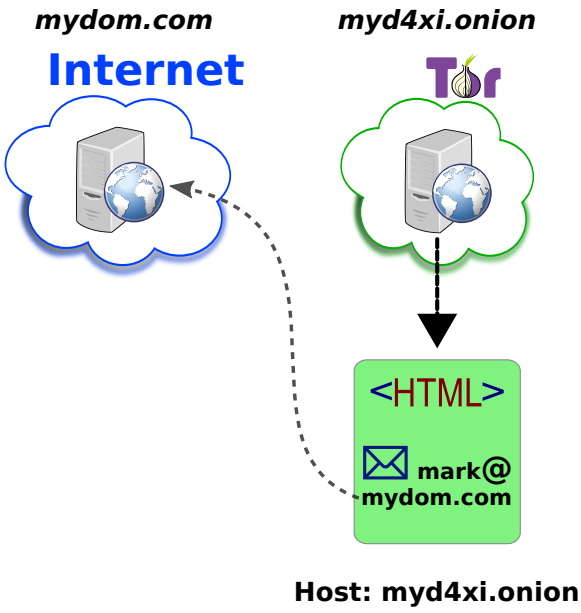

Figure 1.1: Example of a *location leak* in the content accessed by users of a hidden service. The domain embedded in an email address corresponds to a machine, publicly reachable through Internet, that is hosting the hidden service.

be able to leverage those leaks for revealing the real location of the server. When the attacker is a state or an oppressive regime, after the IP address is revealed, the host can be taken down, its content seized, and the owners identified and eventually arrested.

Location leaks are introduced by service owners and for this reason they cannot be centrally fixed by the Tor Project. The Tor Project advertises some guidelines about how to set up your own hidden service [114], but users often reuse content from other applications and it is not uncommon to forget to remove sensitive information. Figure 1.1 depicts an example of a location leak, where a domain reachable from the Internet ("mydom.com") and the hidden service ("myd4xi.onion") identify the same service. Since Tor Network is just an overlay over the Internet, each hidden Web service is hosted on a Web server with a public IP address. In presence of a location leak, once a candidate IP has been identified as the Internet-counterpart of the hidden service, it is possible to connect to it for confirming that is the same host. In the example in Figure 1.1, the owner of the service simply decided to reuse a web page that previously was accessible through "mydom.com". However, before publishing this content, the owner did not sanitize properly all the information contained in the page. The attacker can parse the downloaded document and extract all domains that are included in email addresses. As the next step the attacker tries to connect from

the Internet to each one of the domains that she identified; if the connection is successful and she obtains the same content that was retrieved from the hidden service, the attacker is able to confirm that a candidate domain *is effectively hosting* the hidden service. Analogous problems can arise because of forgotten configuration files that advertise sensitive information (e.g. owner, IP, phone numbers) about the Web server.

Content and configuration leaks are a well-known problem for hidden services, but their extent is currently unknown. Some of the most notorious take-downs have been linked to those leaks: in 2013, in the case of the "Silk Road", the FBI claimed that they located the server because of a leak in its IP address when visiting the site [46]. Despite some incongruities in the official explanation provided by the FBI, security researchers still believe that the take-down was possible thanks to a leak in the configuration of the server [63]. Unfortunately, the same approaches and location leaks that law enforcement might be using to deanonymize abusive services, can also be employed to obstruct freedom of speech: oppressive governments are deeply interested in deanonymizing and censoring hidden services that are linked to political activists and dissidents. If agencies are able to link a service to its owner, the dissident could be condemned to monetary fines or even end up in jail.

The Hidden Service protocol. The architecture of a Tor hidden service comprises the following components:

1. The hidden service.
2. A client, represented by user that has installed the Tor software and wants to access the hidden service.
3. The Rendezvous (Rdp) point is a Tor relay, selected by the client, and used to connect the virtual circuit built by the client with the one built by the hidden service.
4. The Introduction Point (IP) is a Tor relay, selected by the hidden service owner, which is used for forwarding to the hidden service information about the Rendezvous point selected by the client.
5. The Hidden Service Directory (HSDir) is a Tor relay that received the "HSDir" flag. Those relays are stable and long lived and are in charge of storing information about which Introduction Point should the client contact when trying to access a hidden service.

The process of setting up and connecting to a hidden service is depicted in Figure 1.2. Steps ❶-❷ are directly related to the configuration of the service; steps ❸-❽ summarize the process for accessing the hidden service.

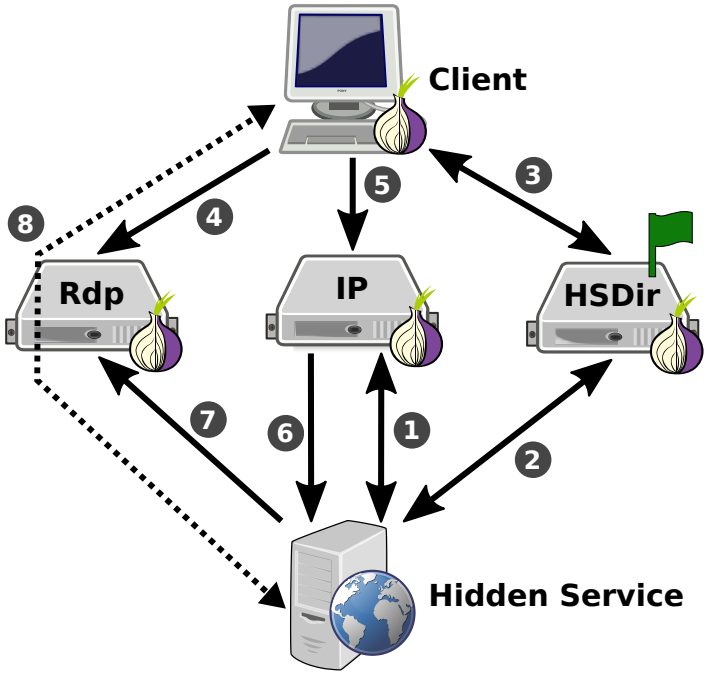

Figure 1.2: Configuration and access to a Tor hidden service.

While setting up a hidden service, the Tor software generates a new RSA key pair. The identifier of the hidden service is the SHA-1 digest of the public key. Then the software selects a small number of Tor relays that will be used as Introduction Points and builds a circuit to each one of them (step ❶). As next step descriptor, containing the ID of the descriptor, the list of Introduction Points and hidden service's public key. Finally this descriptor is uploaded to a responsible Hidden Service Directories (step ❷). The choice of the HSDirs on which to upload the descriptor is performed depending on the ID of descriptor and the fingerprint of the HSDir[2]. Every 24 hours descriptor identifiers are changed in order to change periodically the HSDir responsible for a hidden service.

To connect to a hidden service, the client needs to know the identifier of the service (i.e., its ".onion" unique name). Similarly how the hidden service identified the responsible HSDirs, depending on the identifier the client knows which HSDir it should contact to obtain the descriptor of the service (step ❸). After this the client chooses a Rendezvous point and builds a circuit to it (step ❹). Through the Introduction Point, the client communicates to the hidden service the address of the Rendezvous point that she selected (steps ❺-❻). As a result of this process, also the hidden service builds a circuit to the same Rendezvous point used by the client (step ❼). Finally the client and the hidden service can exchange data

[2]each relay can be identified through a fingerprint, represented by the SHA-1 over its public key

using the Rendezvous as a point of connection of the respective circuits.

1.1.2 Approach

It is possible to run any kind of Internet service as a hidden services, but Biryukov et al. [12] showed that the most popular services include Web and SSH servers. The authors also notice how *"The number of hidden services with illegal content or devoted to illegal activities and the number of other hidden services (devoted to human rights, freedom of speech, anonymity, security, etc.) is almost the same."*. In this dissertation we study the extent of location leaks for Web-based hidden services.

We develop a tool, called CARONTE, that automatically inspects a hidden service looking for possible location leaks that could be used for deanonymizing the service. Location leaks are used to generate "candidate Internet endpoints", which later are checked for verifying if the Internet endpoint corresponds to the Web server hosting the hidden service. CARONTE is composed of the following modules:

- *Exploration*: takes as input "onion" URLs, visits them using different protocols (HTTP, HTTPS), and collects the content of the hidden service and the certificate chain.
- *Candidate selection*: processes the data harvested by the previous module and produces as output a list of pairs in the following format: < "hidden service", "candidate Internet endpoint">.
- *Validation*: after resolving each "candidate Internet endpoint" into a "candidate IP", this module verifies if a candidate IP indeed hosts the hidden services. This is achieved by sending identical requests to the hidden service and to the candidate IP. Then it compares the responses received back: if it finds a pair with similar content and served from a similar web server, it outputs a location leak.

In 2011 Crenshaw proposed a similar approach, which leverages leaks at the application layer for identifying the location of eepSites [22] (the equivalent of hidden services in the I2P anonymity network). I2P is a peer-to-peer network; for this reason the author sets up a node and considers as candidate IPs, that could be hosting a service, all the nodes that belong to the network. This is a so called "closed-world" model, since the adversary already has a list of potential candidates and she only needs to verify if some of them are hosting the service. Tor does not make available an exhaustive list of all nodes that participate in the anonymity network and thus the previous approach does not work in this context. For this reason we significantly differ from Crenshaw's technique, since we operate in an *open-world* model, where a list of possible candidates is not

available. Our approach shows how to move from an open-world to a closed-world by generating candidates directly from the content of the hidden service or HTTPS certificates. This significantly improves our deanonymization results: using the closed-world assumption, and considering as candidates all Tor relays (which have known IPs), 79% of the Hidden Services of the deanonymization achieved by CARONTE could not have been possible. Furthermore the technique proposed by Crenshaw relies only on comparison of server headers obtained from the hidden service and the candidate Internet endpoint. A direct consequence of this is that is possible generate an extremely high number of false positives. We propose instead a more complete approach that performs comparison among both headers and content obtained from the two hosts; this guarantees that during the "validation" phase all-and-only the false positives are filtered out.

1.2 Security Analysis of The Tor Bridge Infrastructure

The Tor Network comprises more than 7,000 routers run by volunteers. Those routers are used for building the virtual circuits through which clients will communicate with their destination, in an anonymous way. Each node of the network is uniquely identified through a *fingerprint*, the 20-byte SHA1 hash of its public key. All nodes are required to speak the "vanilla Tor" protocol and to listen on a dedicated port for connections that use this protocol [27]. *Directory servers* are special routers acting as central authorities and they store the information about all the elements of the Tor Network. When clients need to build a virtual circuit, they will first query directory servers to obtain information about the relays that are available.

The number of daily Tor users grows continuously. Recently Facebook estimated that within a month, over 1 million people connect to their social network through Tor [39]. In countries where governments censor access to particular Internet services [14], Tor is often used for bypassing the filtering systems. This long lived arms race among censors and circumventors led to the development of even more sophisticated systems and techniques. Initially, censors simply blocked access to the Tor Project website [26] or were filtering HTTP requests that had the "/tor/" string embedded in the URL [84]. Furthermore, at its beginnings, all the nodes of Tor Network were relays, so for an adversary was possible to block the access to the whole network: by pretending that she is a legitimate client, she could query the directory server and obtain back the IPs of all relays. To overcome this issue, Tor Project decided to introduce *bridges*, special entry nodes in the virtual circuits, which are not listed in the main Tor directory [113].

When a user decides to run a bridge, the Tor software configures the bridge to automatically send configuration and contact details, including the IP and the port to which to connect, to the Bridge Authority. These files sent to Bridge

Authority are known as "descriptor". Information extracted from descriptors is needed by clients that want to use the bridge. To retrieve information (IP and port) how to connect to a *public bridge*, clients send requests to the Bridge Authority. At any moment of time this authority makes available only a subset of the entire pool of bridges. This strategy prevents an adversary from learning the IPs of all active bridges. When configuring the router, bridge owners can choose among making their bridges *public* or *private*. The only difference among the two configurations is the fact that Tor software running on the bridge will upload or not its descriptor to the Bridge Authority. When the bridge is configured as public, the authority includes it in the pool of bridges that can be given out upon clients requests. On the other hand this does not happen with private bridges: in this case it is the owner that uses an out-of-band mechanism to distribute connection information to the set of users he wants to connect to his bridge.

After the introduction of bridges, it become impossible to know in advance the IP of all Tor routers and for this reason censors started developing more advanced approaches that involve Deep Packet Inspection (DPI). For example, numerous countries as Syria, Iran, Ethiopia, Philippines, Kazakhstan developed techniques that identify Tor traffic by: either looking for Tor's characteristic TLS renegotiation [26] or for a specific Diffie-Hellman parameter used during key negotiation [65], or checking for the presence of particular TLS cipher suites [43, 85, 93]. Currently the most sophisticated approaches are those adopted by the Great Firewall of China (GFC), which uses hybrid approaches that back up an initial DPI inspection, with active probing on the candidate connections [127]. In 2012, after censors started deploying DPI, the Tor Project responded with the introduction of *Pluggable Transports* (PTes) [108]. The idea was to develop a wrapper for the "vanilla Tor" protocol, which should obfuscate the Tor traffic flowing among clients and bridges. Over time several PTes have been proposed. They differentiate because some of them try to imitate popular protocols (e.g. fte [61]), while others encapsulate Tor traffic in popular protocol like TLS (e.g. meek [41]), and others are designed to look as completely random traffic (e.g. Scramblesuit [124], obfs4 [131]). Since censors are turning always more aggressive, design choices for some of these protocols included also protection against active probing (e.g. Scramblesuit, obfs4). A bridge can offer one or more PTs at the same time, the only requirement is that each PT is bound to a different port.

1.2.1 Problem Statement

Even if multiple PTes can be deployed at the same time, bridges are *always* required to be listening on a particular port for connections that use "vanilla Tor". This protocol is a customized version of TLS, with an initial handshake phase, followed by transmission of chunks of data, encrypted using the session key negotiated during the handshake. This protocol was designed to be as close as possible to TLS, in order make Tor traffic harder to spot among other TLS

flows.

Because of a continuous patching process, where all the reported vulnerabilities are fixed by the developers, features like unique cipher list or the characterizing TLS renegotiation, cannot be leveraged anymore by attackers that want to detect Tor traffic [8, 26]. Other ones, like certificate lifetime or unique patterns appearing in the "SubjectCN" and "IssuerCN" of the certificate, are instead still linked to "open tickets" [68]. The reason for this discrepancy is that developers come to the conclusion that efforts required for making "vanilla Tor" indistinguishable from TLS are not a priority, since more secure transports have been introduced as PTes.

In recent years, tools for performing Internet-wide scans in a few minutes have become available. The most famous example is Zmap, a modular open-source network scanner developed at the University of Michigan [31]. Not only tools, but also search engines, that scan on the entire IPv4 address space using a particular port, exist. For example, Censys and Shodan [30, 72], parse the content that was retrieved using a specific protocol on a particular port, and allow users to search for specific keywords or patterns. Both scanning tools and search engines are publicly available and they allow users to access for free the data that was collected. For this reason censors looking for IP addresses that offer one of the protocols supported by bridges, are spoiled for choice. They can choose among free tools, and eventually expand them with the support for the popular protocols used by bridges, or they can query a scan search engine. This has serious consequences for the bridge ecosystem: now not only the attacker does not need anymore to interact with the bridge authority to learn the IPs of public bridges, but by having access to publicly datasets, she becomes able to identify also private bridges.

Also datasets that publish information about the Tor infrastructure, can be abused in a similar fashion. The CollecTor service was launched in 2014 to provide information about nodes and services that are part of the Tor Network [70, 104]. The data that is made available contains information fetched from different nodes that are part of the Tor Network. This data contains also the descriptors collected from bridges. Since those files contain sensitive information, such as the IP and the port to which to connect, before distributing them they undergo to a *sanitization* process. During this step sensitive fields such as email of the owner or the IP are replaced with a bogus value or removed. The final data that is distributed to users is made available in the form of *anonymized descriptors*. The anonymization process affects also bridge *fingerprints*, all the original values are replaced with corresponding SHA1 hash. This stopgap hinders attackers that want to brute-force this value for recovering the original fingerprint, and at the same time owners that know the fingerprint of their bridge can gather historical data. Unfortunately, this is valid also for attackers: if an attacker successfully deanonymized a bridge and obtained its fingerprint, she can abuse CollecTor for ranking the bridge and inspecting its history. This becomes possible, because the

published data includes bandwidth information, the number of clients, counted per protocol or country, that connected to the bridge, uptime, and so on. There is also other valuable information that attackers can extract from bridge descriptors that are available in CollecTor: the *port* on which "vanilla Tor" is running indeed is not sanitized. This gives extremely valuable hints for the attacker: by inspecting the descriptors that are available on a particular day, she can discover what are the most popular ports and then accordingly perform *targeted scan* on those ports.

Bridge owners can deploy multiple pluggable transports on their bridges, with the only condition to bind each PT to a dedicated port. Unfortunately, not all PTes offer the same levels of protection and only the most recent transports are equipped with protection against active probing. This is achieved by requiring clients and bridge to have a shared secret; if the client does not succeed authenticating to the bridge, the bridge will not send back any data to client's requests. Older PTes have weaknesses [37] that can be exploited for identify particular PTes with a negligible number of false positives. Furthermore works have shown how difficulties of supporting all the features of the protocol the PT is trying to mimic, can be used for deanonymizing bridges with high confidence [51, 119]. In case multiple PTes, with different levels of protection are available on a bridge, the attacker can focus on the *weakest protocol* for verifying that particular host is a Tor bridge. The censor that we consider is a state-level entity that wants to identify communications with Tor bridges for monitoring or blocking the traffic [37, 100, 129]. This powerful adversary has access and *full control* over the data flowing from the monitored network to Internet; thus she can tamper with the traffic, drop and inject packets or even impersonate legitimate Tor users trying to connect to the Tor Network. Finally, this adversary can also perform Internet-wide scans and query public public information sources as search engines scan search engines or CollecTor.

1.2.2 Approach

In this section we discuss the steps that an adversary needs to take in order to deanonymize bridges, how she can evaluate how successful she is and how she keeps track of address changes of bridges.

The first, essential, step of the entire process is an Internet-wide scan on a particular port, followed by the collection of the TLS handshake for those hosts that had the port open. The unique patterns used for "SubjectCN" and "IssuerCN" that appear in each Tor certificate guarantee that any IP found serving those certificates has to be a Tor router. The CollecTor data is a useful guidelines for performing targeted scans on the most popular ports, since it contains information about the OR port on which routers are offering "vanilla Tor" protocol. Otherwise, the adversary can also decide to opt for a lightweight approach and to do not perform any scans on his own and simply query existing search engines

such as Shodan or Censys [30, 72].

As the next step the adversary has to distinguish relays from bridges. Since the data published by CollecTor contains non-sanitized descriptors for relays, they can be used for generating a list of all the IPs linked to relays. Any other IP, that is serving a Tor-alike certificate and does not appear in the list of relays, is a bridge.

Thanks to the previous two steps the adversary has successfully identified all the addresses that were bridges when the TLS handshake was collected. Now she can verify that that the candidate IP that was identified is still hosting a bridge. During this phase, she uses the "vanilla Tor" protocol to connect to a deanonymized IP and collect the non-sanitized descriptor directly from the bridge. This third optional step on one hand guarantees that a bridge is still running on the IP that was identified, on the other one it is needed for obtaining the bridge's descriptor. Later on, from the descriptor the adversary can extract contact information about the owner of the bridge or the fingerprint.

Hashing the fingerprint allows the adversary to obtain historical data about the bridge, just by searching in the CollecTor data for a particular sanitized fingerprint. This is not useful only for performing a ranking of the deanonymized bridges, but also for deciding upon the type of a bridge. If the value for the sanitized-descriptor is available in CollecTor, the bridge is *public*. In case of *private* bridges, the adversary learns that any Tor users connecting to a private bridge could be a member of the same organization or group, and she can geographically locate them.

Ranking deanonymized bridges is an extremely important step for the adversary because it is the only way to answer fundamental questions: "what coverage am I achieving?" and "did I identify all the bridges that are available for my country?". To evaluate her deanonymization coverage, the adversary extracts information from the CollecTor data and ranks bridges with respect to their importance according to different metrics. For example, the adversary can decide to focus only on bridges that are popular in a particular country, or on bridges offering high bandwidth and that have the highest number of users.

In case bridges are linked to dynamic addresses, the adversary needs a way for keeping track of the bridge each time it changes its IP. To this end, after she has identified the IP of a bridge, she can perform a *vertical scan* on all ports on that host. The goal of the adversary is now to enumerate other known services that are offered by that address. If a service has a unique identifier (e.g. SSH keys, certificate identifiers), this information can be used for keeping track of address changes. Shodan regularly scans over 200 ports with popular protocols and indexes the information that it collects. The unique identifier obtained from a bridge can be used in Shodan to generate the list of IPs that were found serving that identifier; then the adversary will connect to each candidate using "vanilla Tor" protocol and confirm if it is a bridge.

1.3 Thesis Contributions

In this dissertation we design and evaluate techniques for identifying the IP address of hidden servers and use them to analyze the security of two kinds of hidden servers in the Tor anonymity network: hidden services and bridges. Despite our experiments being conducted on Tor, several of our techniques and findings can be generalized to other anonymity networks. This dissertation comprises the following works:

I. **CARONTE: Detecting Location Leaks for Deanonymizing Tor Hidden Services**
Srdjan Matic, Platon Kotzias, and Juan Caballero
Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, USA, October 2015.

II. **Dissecting Tor Bridges: a Security Evaluation of Their Private and Public Infrastructures**
Srdjan Matic, Carmela Troncoso, and Juan Caballero
Proceedings of the Network and Distributed System Security Symposium, San Diego, CA, USA, March 2017.

An overview of the contributions of the first work is available in Section 1.3.1. The full description including implementation, technical details, and discussion is available in Chapter 3. An overview of the contributions of the second work is available in Section 1.3.2. The full description including implementation, technical details and discussion is available in Chapter 4.

1.3.1 CARONTE: A Tool For Deanonymizing Tor Hidden Servers

We design, implement, and evaluate techniques for automatic recovery of the IP address of Tor Hidden Services. Our approach extracts leaks from the application layer of a HS, and uses them for trying to identify the location the machine hosting the service. To this end, our work makes the following contributions:

- We propose a novel approach to deanonymize hidden services through location leaks, extracted from the application layer of a HS. Our approach assumes an open-world model, i.e., no prior knowledge of candidate servers where the HS could be hosted. To move from an open-world to a closed-world we propose techniques to identify candidate servers directly from the content and configuration downloaded from the hidden service. Our approach comprises a validation step for eliminating possible false positives.

- We implement our approach into CARONTE, a tool that automatically attempts to deanonymize the location of hidden services through location

leaks. CARONTE can be used by service owners to check to perform auditing of their HS, or by law-enforcement agencies that want to identify the location of a particular hidden service.

- Using CARONTE we perform the first measurement study on the prevalence of location leaks in hidden services. CARONTE analyzes 1,974 input hidden services, successfully recovering the IP address of 101 (5%) of them. It also uncovers that 21% of the deanonymized hidden services are running on Tor relays.

1.3.2 Security Analysis of the Tor Bridges Infrastructure

We perform the first systematic study of the public and private bridge infrastructure in the Tor Network from a security point of view. We study security-relevant properties of bridges using publicly available datasets such as Collector [104] and scan search engines. To this end, our work makes the following contributions:

- We perform the first in-depth study of the security implications of the data publicly available through CollecTor. We demonstrate how an adversary can leverage CollecTor to measure the coverage of its blocking and to select OR ports to scan next in order to disable highly valuable target bridges. We also show that the adversary can measure security-relevant bridge properties such that default bridges are used by bulk of users (despite being trivial to deanonymize), and that bridges carrying users are stable and rarely change their IP address, and thus can be blocked for a long time.
- We identify a security issue in the deployment of Pluggable Transports where bridges offer a mix of PTes with different security properties. For example, the most recent PTs include reply protection, which guarantees that the bridge will allow connections, and transmit data, only for users that are able to authenticate. We identify several bridges mixing older and newer protocols with and without this kind of protection. This enables an adversary to identify the bridge through the weaker PT and disable the stronger PTes by blocking its IP.
- We present the first study of private bridges that are not known by the Tor Network, nor appear in Collector. Private bridges comprise 35% of all bridges we discover and deanonymize.
- We propose a novel technique for the adversary to track bridges across IP address and OR port changes by leveraging that hosts running bridges can be running also additional service, which may provide trackable unique identifiers.
- We identify a previously unreported element of the Tor Network represented by proxies, i.e., private IP addresses that forward traffic to a backend bridge or relay. We study the infrastructures built using proxies and bridges using

a novel clustering approach to group ORs owned by the same entity based on their configuration and IP addresses.
- We provide an extensive security discussion on the security implications of our findings, summarizing all the issues that we identified and we propose possible counter measures.

1.4 Ethical Considerations

While there exist tools like Shadow [57] to simulate attacks against Tor, they cannot be used in our work. The application layer leaks we studied, cannot be easily simulated because they are caused by erroneous configurations by administrators of hidden services. Analogously, any tool for simulations cannot be used neither for assessing the security of Tor bridges, since we use public data repositories that contain information about elements of the network. Similarly to Biryukov [13] we deem experiments on the live Tor network worthwhile and necessary to enhance the scientific understanding of hidden service and the anonymity network itself, as long as those experiments to not cause degradation of the network or its services.

To deanonymize the hidden services that we analyzed, we did not take into account possible software vulnerabilities, despite often, just by by inspecting some content generated automatically, we were able to identify the software that was used and look for vulnerabilities in case outdated or unpatched versions of a particular software were involved. We decided to do not explore this approach because even if we were able to correctly identify the software and the version used by a specific service, the only way that we could gather information about the location of the server would be by effectively *exploiting the vulnerability*. All the information that we collected is obtained through legitimate interaction with the service and without forcing any unwanted behavior of the application. All the certificates and content have been downloaded because the service was explicitly configured to provide those files upon a request to a web page. Finally, during the exploration of hidden services, to prevent downloading copyrighted material and offensive content (e.g., pornography), we limit our tool to collecting textual and HTML content, ignoring other content such as images, videos, or documents.

The bridge deanonymization techniques we propose only leverage known limitations of Tor Network, and only use leaks present in publicly accessible repositories, such as Shodan, Censys or CollecTor. We purposefully avoid adding relays or bridges into Tor Network, as well as exploiting any software vulnerability. We have no access to any traffic that is not ours, and hence we can no threaten the privacy of any Tor user. However, the data we collect contains the IP addresses and contact information of both public and private bridges that must be kept private to preserve the security provided by Tor Network. Thus, we do not disclose any bridge/proxy/hidden service IP address, nor any personal information

we may learn about its owners, but we illustrate important steps and findings only through aggregated data and fake examples.

All works have been approved by by the ethics review board of our institution, which has mandated that due to its sensitive nature all the data must be protected with diligence, must not be be disclosed to third parties, and must be deleted when the paper is published.

After each submission, we have sent a copy of this draft to Tor Project to notify them of the work. Specifically in the case of the work about Tor bridges, The Tor Project already started taking countermeasures [69] to prevent bridge targeting based on CollecTor public information.

2

Related Work

Interest toward anonymous communication and services is a topic that has been explored in academia for several decades. The first work toward this direction dates back to 1981, when David Chaum presented "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms" where the author discusses features and possible implementation of an anonymous email system [17]. His technique was based on cryptography and allowed the correspondents of email messages to remain anonymous to each other while communicating via untraceable return addresses. This could be achieved through *mixes*, intermediate nodes in charge for reordering, padding and delaying users' data in order to complicate traffic analysis. Following analogous research questions, in 1996 Goldschlag et al. [44] introduced the concept of *onion routing*. The proposed approach separates the anonymity for the connection, from the anonymity of the communication that is flowing over the connection. Onion routing suited for real-time communication and would protect Internet services against passive and actives attackers attempting to eavesdrop the communication or perform traffic analysis attacks. To this end the authors leveraged a layered data structure called *onion*, that relies upon IP routing to reach the destination. According to a similar naming convention, routers in charge of forwarding these data structures are called *onion routers*. The anonymous connection is obtained by chaining together a sequence of onion routers from an initiator to the responder, in a way that each router can only identify the previous and the next hop along the route. Furthermore, the data that is sent over the anonymous channel is encrypted with several layers of encryption, and as it moves through the anonymous connection, each onion router removes one layer of encryption such that what is delivered to the responder is only the plaintext. Independence from the application layer and real-time constraints were respectively both pros and cons of the onion routing compared to Chaum's work. The separation from the application layer offered better flexibility and more use cases for the anonymous channel. On the other side, requiring to deliver data in real-time limited considerably the freedom in mixing. In 1998

Reed et al. [90] implemented a prototype with the specifications of onion routing. The prototype allowed remote login requests, offered possibility to proxy HTTP requests and exchange electronic emails.
By keeping unknown the physical location of a website it becomes possible to protect the service from seizure orders. This design principle is the core of the storage medium service that Anderson [3] designs for offering resistance to denial of service attacks. The proposed system leverages redundancy and scattering techniques to replicate data across several machines. Anonymity is then used to protect this information from selective service denial attacks. A similar idea is also the basics for the design of hidden services, as proposed by Dingledine et al.in 2004 [28]: a service for which the location is unknown can resist to censorship, physical and Dos attacks [25]. Starting from that moment several works focused on breaking the sender-recipient anonymity achieved using virtual circuits and rendezvous points [24].

The first work toward this direction was proposed by Øverlier and Syverson [86] who show how a hidden service could be rapidly located because the HS selects nodes at random when building connections. By having control over a single node of the network, the attacker can report a false higher uptime and maximum network bandwidth, in order to be selected by the HS. Once an attacker's node is the one closest to the hidden service, by correlating input and output traffic, the attacker can discover the effective IP address of the server. Bauer et al. [10] extended Syverson's attack to general purpose circuits. They study possible impacts of Tor routing optimizations on the ability to provide strong anonymity. The authors demonstrate how a lying adversary, that simply exaggerates on her resource claims, can compromise an unfair percentage of both entry and exit nodes. By advertising an extremely high bandwidth and uptime, the adversary induces clients to select her nodes for building circuits. An attacker controlling entry and exit node, can then compromise the anonymity of a path even before any data is transmitted.

The solution for the attacks mentioned above was the introduction of *Entry Guard Nodes* as proposed by Wright [130]. This guarantees that the hidden service will always use the same node, or a small subset of trusted nodes, as the first node in a communication. Elahi et al. [34] introduce improvements to reduce the guard compromise rate, developing a simulation-based research framework that is used for studying Tor's entry guard design. With the support of their framework, the authors demonstrate how short term entry guard churn and explicit time-based entry guard rotation contribute to clients using more entry guards than they should, and this situation increases the likelihood of profiling attacks. In 2013, Biryukov et al. [13] are the first ones to perform a large scale study on the hidden services at that time. The authors show how an attacker can exploit design and implementation flaws both in the design of hidden services to measure the popularity of a particular service and make the service unreachable. This is achieved by injecting malicious relays in the Tor Network, and forcing

them to be selected as the hidden server directories. Furthermore, the authors also illustrate how an attacker can leverage correlation attacks for both verifying if her relay was selected as the guard node of a hidden service, and for identifying the guard node in case the relay that belongs to the attacker is used as a middle node.

The attacks previously mentioned rely both on vulnerabilities in the Tor specification and on adding malicious relays to the network. Our approach differs from the previous techniques since it does not require the attacker to add nodes to the Tor Network and leverages only location leaks for large-scale deanonymization of hidden services. Since those leaks are introduced by service owners, they can not be centrally addressed by the Tor Project.

Project OnionScan [95], was started in April 2016 and leverages on a approach similar to ours for identifying possible information leaks within hidden services. The proposed tool is able to identify a broader class of leaks that the one of which we focus and that are useful only for recovering the IP address of the server. PGP identities or metadata obtained from images can provide information about owners of the service. Unfortunately these kind of leaks are hard to validate automatically and might require investigations or even manual police work. More in general, OnionScan limits it capabilities to leak detection and does not have a rigorous validation step as the one that we propose. Our validation step instead ensures that each candidate IP address that was identified, is effectively hosting the hidden service that we were looking for. Finally, compared to OnionScan, we also identify candidates and deanonymize hidden services using information from TLS certificates. On the other side, OnionScan supports other protocols in addition to HTTP and HTTPS.

Another approach similar to ours also was proposed by Crenshaw et al. for deanonymizing I2P eepSites [22]. Crenshaw sets up an I2P router and uses the IP addresses of the peers known to his router as candidates for hosting the eepSites. This approach works for the closed-world model of I2P, where it is possible to know in advance the list of all the nodes participating in the network. In Tor, which is not peer-to-peer based, this approach is similar to considering the list of all Tor relays as candidates for hosting a hidden service. Using this closed-world approach we would be able to deanonymize only 21% of hidden services that we deanonymized. Our approach in contrast shows how to move from an open-world to a closed-world scenario, by obtaining candidates from the content and configuration downloaded from the hidden services.

Other closed-word approaches use clock information for deanonymizing hidden services. Murdoch [80] investigates how clock skew changes can be leveraged for recovering the IP address of a hidden service. The attacker accesses the hidden service and vary traffic over time in order to cause the server to heat up or cool down, according to a specific pattern. The list of candidates available to the attacker, is then used to verify if any of them is hosting the service that she is trying to deanonymize. The presence of load pattern induced in the hidden service,

observed in one of the candidates, is used to verify that the candidate is hosting the service. In his work, Zander et al. [132] improve the previous technique to require less network traffic and achieve higher accuracy. The extended approach estimates the clocks skew directly from the anonymous channel and works with low-frequency clocks.

Several authors explore also the risks of deanonymization attacks that leverage DoS. Jansen et al. [58] propose a low cost attack against Tor, that an adversary can use to disable anonymously any Tor relay. The idea is to build a circuit and then to stop reading that data that is sent over the circuit. In this way, the target relay will keep buffering cells until it runs out of memory. Tests on live Tor network proved how, after only half of an hour, the attack successfully disabled all of the top 20 exit relays, reducing overall Tor's bandwidth by 35%. The approach proposed can be extended also to deanonymize hidden services. To this end, it first requires to identify the guards of a hidden service, for example following the technique proposed by Biryukov et al. [13] and then to systematically disable the guards. As a final step, the attacker can test if the hidden services selected one of her malicious relays as a guard node. A similar method relying on DoS is proposed also by Borisov et al. [16]. In this case, the authors investigate the dangers of selective DoS attacks, where attackers aim at compromising reliability of the system in those states that are hardest to compromise. This approach has a twofold advantage, on one hand it is much cheaper than a DoS on the entire system, and on the other the attack could cause the system to enter in less secure states. Faced with poor reliability, many users (and a lot of software solutions) will attempt the communication again, presenting more opportunities for attack. As a countermeasure, Øverlier and Syverson [87] propose Valet Service nodes to improve the resilience by introducting points against DoS attacks.

A separate line of work considers the forensics problem of proving that a confiscated machine hosted a hidden service [36, 96]. Those approaches propose to leave a timing channel fingerprints in the confiscated machine's log file. These fingerprints are later used as an evidence that the server was hosting a particular content.

Syverson et el. [99] study how an attacker, able to observe encrypted traffic from a client to the first relay and from the final relay to the destination, can link the client his destination by correlating traffic patterns. Other more general works, show how powerful adversaries that can establish themselves in that position both at the Internet exchange [82] and the AS [2, 33, 40, 59, 130] levels, also represent a serious threat for users' anonymity.

Finally, several works propose traffic analysis attacks for linking the initiators of connections with their respective communication partners and vice versa. Murdoch and Danezis [81] show how an attacker can identify which relays are on a target Tor user's path, by building paths one at a time through every Tor relay and introducing congestion. Evans et al. [38] improve the previous approach, by introducing a novel bandwidth amplification attack. Hopper et al. [50] show the

amount of information that can be inferred using round-trip time and how malicious web sites that receive multiple connections from the same exit node, can predict if the two connections are using the same Tor circuit. Mittal et al. [75] use traffic analysis to identify guard relays of users and to break unlinkability by determining if two Tor connections belong to the same user.

One of important features Tor is known being used for is to bypass censorship [103]. Generally this is achieved first by connecting to a bridge and then by requesting the forbidden content through the Tor Network. For this reason academic community has dedicated also much work and attention to Tor bridges and their circumvention capabilities. One of the research lines, deals with proposal of new Pluggable Transports to protect the communication from attacks that rely on traffic analysis [32,76,120,128]. Weinberg et al. [120] develop a novel encrypted transport protocol which is implemented in a tool called StegoTorus. The idea is to leverage steganography and to hide Tor traffic within an innocuous covert protocol such as HTTP. To foil analysis of packet contents, traffic originated from the Tor circuit is distributed over many shorter-lived connections with per-packet characteristics that mimic the cover-protocol traffic. Moghaddam et al. [76] propose a new technique for embedding messages, sent to Tor bridges, in a protocol that is widely adopted over Internet. To this end they develop a prototype for of a Pluggable Transport that encapsulates Tor traffic in Skype video calls. To morph Tor streams into Skype video calls, the authors propose two different methods that use traffic shaping for emulating the distribution. Winter et al. [124] design and implement Scramblesuit, that relies on protocol obfuscation for hiding the Tor traffic. By using a secret previously exchanged out-of-band, ScrambleSuit can defend against active probing and other fingerprinting techniques such as protocol classification and regular expressions. Studies for techniques that can be used for circumventing DPI technologies include also the work by Dyer et al. [32]. The authors develop a tool able to bypass both enterprise and governments DPI systems, using format-transforming encryption. This technique provides a generic and flexible mechanisms for turning arbitrary application-layer protocol messages into ciphertexts that DPI will classify as messages from a known protocol.

In this same research track we can include also works that focus on detection of the new Pluggable Transports that were proposed. Wang et al. [119] provide an in-depth investigation of the detectability of in-use protocol obfuscators by DPI. They show how novel attacks that leverage entropy information and machine learning can be used for detecting obfuscated traffic. They first build a framework that is then tested on live network and with five different Pluggable Transports, and they show how their framework generates a sufficiently low number of false positives to be used in many censorship settings. Houmansadr et al. [51] show that Pluggable Transports that mimic other protocols, e.g., [76, 120], fail to achieve unobservability. Since those system mimic only part of the communication, the authors provide dozens of passive and active methods that recognize even a single imitated session.

A second line of work focus on the approaches censors use to confirm that suspicious nodes are bridges. Wilde [121] investigates the active probing mechanism Great Firewall of China adopts to identify Tor connections. The author shows how DPI performs probes in near-real-time right after a Tor connection, including a TLS handshake on port 443, happens. Furthermore, his analysis indicates also that the Tor TLS cipher list turned out being a significant component for the detection mechanism that triggers the active probing. Winter et al. [126] perform an exhaustive investigation on the blocking mechanism that are applied. To this end they evaluate the policies which are used for blocks, the location of the scanners that the firewall is using to identify bridges, the frequency on which scanners connect as well as attempts to use spoofed IP addresses. Finally the authors discuss how the firewall can be bypassed using packet fragmentation. Further details about the censorship mechanisms used in China is disclosed by Ensafi et al. [37]. Their work focuses on identifying the different types of probing, the development of fingerprinting techniques to infer the physical structure of the system and the localization of the sensors that trigger probing. The authors test probers' efficacy using different versions of Tor and their results show how the censoring system is updated regularly, it operates in real time and that blocks to Tor nodes are lifted every 25 hours.

Finally, another research line is dedicated to the discovery of bridges. McLachlan et al. [74] investigate weaknesses of the current bridge architecture and show that, when bridges run in clients, it is possible to deanonymize bridge operators and that Tor exit nodes can be used for enumerating a larger set of bridges. Ling et al. [66] propose a different active approach, where the attacker can either directly interact with the distributor, and obtain the addresses of bridges through email, or she can set up a relay and passively that bridges connect to her node. Their test on the real network proves how after only a few weeks is possible to collect information about thousands of bridges. However, main limitation of these works is fact the distributor advertises only a small subset of all bridges that are available. Furthermore, since the proposed technique relies also on the introducing of a relay controlled by the attacker, this step requires first to passively wait for bridges to build a circuit to the malicious node, before being able to identify them.

3

CARONTE : Detecting Location Leaks for Deanonymizing Tor Hidden Services

This chapter reproduces the work "CARONTE : Detecting Location Leaks for Deanonymizing Tor Hidden Services" published at CCS 2015. This paper shows a new and general approach for deanonymizing hidden services in an open-world scenario leveraging only location leaks. This work was realized together with people from the IMDEA Software Institute.

3.1 Introduction

The increasing surveillance of communications have made anonymity networks a critical privacy-enabling technology. Tor [28] is arguably the most popular anonymity network. It provides both sender anonymity and recipient anonymity for *hidden services*. Hidden services protect the location (i.e., IP address) of the server hosting the hidden service. In addition, they further protect against network-level eavesdropping by providing encryption all the way from the client to the hidden service. This includes the communication between the last Tor relay and the hidden service, even when the application traffic is not encrypted.

Deanonymizing the location of a hidden service, i.e., recovering the IP address of the server hosting the hidden service, enables taking down the hidden service, seizing its content, and possibly identifying the owners. Prior work has proposed attacks to deanonymize Tor hidden services through flaws on the Tor protocol [13, 86] and clock-skew fingerprinting [80, 132]. Attacks on the Tor protocol are promptly fixed by the Tor Project. For example, the attack by Øverlier and Syverson [86] was fixed by introducing guard nodes and the more recent attack by Biryukov et al. [13] has also been fixed [12]. Attacks leveraging clock-skew fingerprinting assume a *closed-world* where a short list of possible candidate servers is known and the fingerprinting validates which candidate is the hidden server. Deanonymization attacks have also been proposed for the equivalent of hidden

services in I2P (called eepSites) [22]. These attacks also assume a closed-world, where the IP addresses of I2P peers are candidate servers for eepSites.

This paper studies the problem of *location leaks*, i.e., information in the content or configuration of a hidden service that gives away its location. Location leaks are introduced by the hidden service administrators and cannot be centrally fixed by the Tor Project. Deanonymizing hidden services through location leaks does not require the attacker to be part of the anonymity network, but only to access the hidden services.

Such content and configuration leaks are a well-known problem for hidden services, but their extent is currently unknown. In fact, some notorious takedowns of hidden services by law enforcement have been linked to such leaks. For example, in July 2013, law enforcement identified the location of the Silk Road marketplace, where products such as cocaine, heroin, LSD, and counterfeit currencies were traded [21]. The FBI claimed in court that they located the server of the original Silk Road through a leak of its IP address when visiting the site [46]. The FBI story has been disputed [46, 63], but researchers still believe it is likely that the takedown was due to a leak in the server's configuration [46]. After the Silk Road takedown other similar hidden services took its place. In November 2014, an international law-enforcement operation codenamed *Onymous*, took down over 400 hidden services including the Silk Road 2.0, Cloud 9, and Hydra drug markets [47]. The deanonymization method used by law enforcement in Onymous remains unknown [109]. Unfortunately, the same location leaks that law enforcement may be using to deanonymize abusive hidden services can also be used by oppressive governments to deanonymize and censor hidden services of political activists.

In this paper we propose a novel approach to deanonymize hidden services using a subset of location leaks in an *open-world*, i.e., without previous knowledge of a set of candidate servers. Our approach includes two steps. First, we propose techniques to extract candidate Internet endpoints (i.e., domains and IP addresses that may correspond to the hidden server) from the content and configuration of a hidden service. Our techniques examine endpoints and unique identifiers in the content and the HTTPS certificates. This step allows moving from an open-world to a closed-world. Then we validate each candidate pair $\langle$ *hidden_service, Internet_endpoint* $\rangle$, checking if the Internet endpoint corresponds to the Web server hosting the hidden service. Previous work that leverages leaks on a server's clock skew [80, 132] or software version [22] assume a closed-world and use the leaks only for validation. In contrast, our approach leverages location leaks to obtain also the candidate servers.

We implement our approach in a tool called CARONTE, which takes as input the URL of a hidden service and tries to deanonymize it through location leaks. CARONTE could be used by law enforcement agencies to automate the currently manual process of deanonymizing abusive hidden services, and also by political activists presumably so they can detect and remove potential leaks that risk

being censored. CARONTE checks a hidden service for a subset of content and configuration errors that can lead to deanonymization. CARONTE has no false positives. If it recovers the hidden service's IP address, it proves the need to improve operational security. However, it only tests for a subset of location leaks whose detection can be automated and may fail to validate some leaks. Thus, it cannot guarantee the hidden service is free of location leaks.

To test CARONTE's effectiveness we have applied it to 1,974 live HTTP and HTTPS hidden services, of which CARONTE recovers the IP address of 101 (5%). Our results can be considered the first measurement study of location leaks in Tor hidden services. Since CARONTE only tests for some types of location leaks and may fail to validate some leaks its results are conservative, i.e., some services not deanonymized could still be vulnerable.

Our results also show that 21% of the deanonymized services are hosted on Tor relays. Hidden services on Tor relays can easily be deanonymized, even in the absence of location leaks, assuming a closed-world where relays' IP addresses are candidate locations for a hidden service. This result also highlights the importance of assuming an open-world, as 79% of hidden services CARONTE deanonymizes cannot be deanonymized under the closed-world assumption. CARONTE also identifies 9 hidden services that redirect their users to Internet sites through HTTP, negating the benefit of encryption in the last hop.

This work makes the following contributions:

- We propose a novel approach to deanonymize hidden services through location leaks. Our approach assumes an open-world, i.e., no prior knowledge on candidate servers. To move from an open-world to a closed-world we propose techniques to identify candidate servers from the content and configuration of a hidden service.

- We implement our approach into CARONTE, a tool that attempts to deanonymize the location of hidden services through location leaks.

- Using CARONTE we perform the first measurement study on the prevalence of location leaks in hidden services. CARONTE analyzes 1,974 input hidden services, recovering the IP address of 101 (5%). It also uncovers that 21% of the deanonymized hidden services are running on Tor relays.

The rest of this paper is structured as follows. Section 4.2 introduces hidden services, location leaks, and our approach. Section 4.4 details the approach and CARONTE's implementation. Section 3.4 presents the measurements. Section 3.5 discusses defenses against location leaks and Section 4.10 ethical considerations. Section 4.9 describes related work and Section 4.11 concludes.

3.2 Overview and Problem Definition

In this Section we first briefly describe hidden services (Section 3.2.1), then we detail the type of leaks that CARONTE looks for (Section 3.2.2), and finally we provide an overview of our approach (Section 3.2.3).

3.2.1 Hidden Services

The Tor network provides hidden services as a mechanism for users to anonymously offer services accessible by other users through Tor. Hidden services provide recipient anonymity, i.e., they hide the IP address of the server hosting the hidden service. Hidden services can offer different functionality, e.g., Web services and Bitcoin mining. This paper focuses on hidden Web services.

The creation of a hidden service picks a public key and a secret key and creates an onion identifier by doing a SHA1 hash of the public key truncated to 80 bits. The onion identifier is encoded in base32 producing a 16 character string, which is appended the suffix ".onion" to produce an *onion address*, e.g., `niazgxevgzlrbpvq.onion`[1]. Some hidden services generate large numbers of onion addresses until they find one containing a desired substring. For example, Facebook's official hidden service has onion address `facebookcorewwwi.onion`. For Web services, hidden services are advertised as *onion URLs*, where the DNS domain is replaced by an onion address, e.g., `http://niazgxevgzlrbpvq.onion/content.html`. The distribution of onion URLs happens out-of-band: there is no central repository that lists all the hidden Web services available in a given moment and thus users can access only those for which they know an onion URL.

Rendezvous protocol. The hidden service chooses a set of 3 Tor relays as *introduction points* and creates *circuits* (i.e., encrypted tunnels) to each of them. The hidden service then produces a signed *descriptor* that lists the service's public key and its introduction points. The descriptor is published in a distributed hash table on the Tor relays using as index the onion identifier and time period.

To use a hidden service a Tor client establishes a circuit to a Tor relay randomly chosen as *rendezvous point*. Then, it retrieves the hidden service's descriptor using the onion identifier and current time. Next, it creates another circuit to one of the introduction points, and communicates the rendezvous point to the hidden service through it. The hidden service creates a circuit to the rendezvous point and communication between client and hidden service happens over their respective circuits to the rendezvous point.

With hidden services all hops between client and hidden service are encrypted, compared to visiting an Internet domain through Tor where communication between last Tor relay and the service is not encrypted, unless the application layer is encrypted (e.g., with HTTPS).

[1] Fake onion address for illustration.

3.2.2 Location Leaks

Hidden services hide the location of the server hosting them, but they do not protect against administrators unintentionally leaking information in the content or configuration of their hidden services that may lead to deanonymizing them (*identity leaks*) or the IP address of the hidden server (*location leaks*). The Tor Project describes the risks of content and configuration leaks [114] but it is not clear to what extent hidden services administrators are taking precautions against them and how prevalent they are.

Using a leak in the content or configuration of a hidden service to deanonymize their administrators or its location involves two steps. First, find some *candidate identity* (e.g., the owner of a phone number embedded in the hidden services content) or *candidate Internet endpoint* (e.g., an IP address or DNS domain in an error page). Then, *validate* that the candidate identity truly corresponds to the administrators or the candidate Internet endpoint to the IP address. In this work we focus exclusively on location leaks that provide candidate Internet endpoints for the IP address of the hidden service, by simply visiting the hidden service, without adding any nodes to the Tor network or compromising the hidden service. Identity leaks as well as leaks that an attacker can induce by exploiting the hidden service's code (e.g., SQL injections) are considered out of scope.

We do not focus on location leaks because they are more important, but rather because we know a way to automatically validate candidate Internet endpoints to confirm that they are hosting the hidden service. Validation of identity leaks is better suited for manual police work than for an automated tool. Of course, deanonymizing the location of a hidden service may be a first step towards deanonymizing its owner's identity, e.g., by checking server registration records or monitoring accesses to the server. Note that location leaks are a problem for any type of anonymous services. While we focus on Tor hidden services, our approach is independent of Tor and can be applied to deanonymize the equivalent of hidden services in other anonymity networks, e.g., I2P eepSites [55].

There exist many different types of location leaks that can lead to deanonymizing the hidden server's location. CARONTE automatically identifies 3 types of location leaks due to *endpoints* (i.e., IP addresses, domains) and *unique strings* (i.e., Google Analytics ID, Google AdSense ID, Bitcoin wallets, page titles) embedded in the content of the hidden service, and the *HTTPS certificate* of the hidden service. Obviously, many other types of location leaks may exist and thus CARONTE may not find all location leaks.

Validating location leaks. Our candidate selection techniques output candidate pairs $\langle onion\_address, Internet\_endpoint \rangle$, where the Internet endpoint is a DNS domain or IP address that may host the hidden server of the onion address. If the candidate pair contains an IP address, validation connects to the IP address through the Internet but requests the content of the hidden service,

i.e., sets the HTTP Host header to the onion address. If the Web server delivers the hidden service's content, it confirms that the hidden service is hosted on the candidate IP address. Note that the Tor network is an overlay of the Internet. Thus, hidden Web services are hosted on a Web server with a public IP address. The (hidden) Web server will reply to a request from the Internet to its public IP address unless a firewall allows only connections from certain IP addresses. Even when a firewall is used, the firewall will often be configured to allow connections through Tor (i.e., from Tor relays), in which case we can run validation through Tor (or from a Tor relay).

When the candidate pair contains a DNS domain, validation works the same by first resolving the domain to an IP address. The main reason why an Internet domain may resolve to the public IP address of the hidden Web server is when the domain identifies an Internet service hosted on the same Web server as the hidden service. Thus, if the hidden service runs on its own Web server CARONTE will probably fail to validate a candidate domain, even if the candidate domain belongs to the hidden service owners. For example, CARONTE could identify from the content of the hidden service an Internet site owned by the hidden service's owners, but hosted on a separate Web server. While CARONTE cannot validate that location leak, network-level attackers could monitor connections to the Internet site the hope that the hidden service owners relax protections when accessing it. For that reason, CARONTE still outputs the candidate domains it found even if they are not validated, so that the hidden service administrator can check them.

In summary, after the final validation step, CARONTE does not generate false positives since all leaks were inspected and confirmed. All leaks output after the validation step, indicate not just that CARONTE successfully downloaded the same content from the hidden service and the Internet Endpoint, but also that the two servers are co-located on the same physical machine. On the other hand, CARONTE may have false negatives due to unsupported classes of leaks and leaks it cannot validate. We discuss how to safely configure hidden services in Section 3.5.

Unintentional leaks. Hidden services often contain sensitive content and need to protect their location to avoid censorship or legal prosecution. However, not all hidden services are like that. Some hidden services advertise an Internet clone, i.e., an Internet site that provides the same content, and explicitly link to it, e.g., Facebook. Similarly, an Internet site may explicitly link to its hidden service clone, indicating its availability to privacy-concerned users. In Section 3.4 we describe 3 automatic checks that CARONTE performs to classify location leaks leading to deanonymizations as unintentional or not.

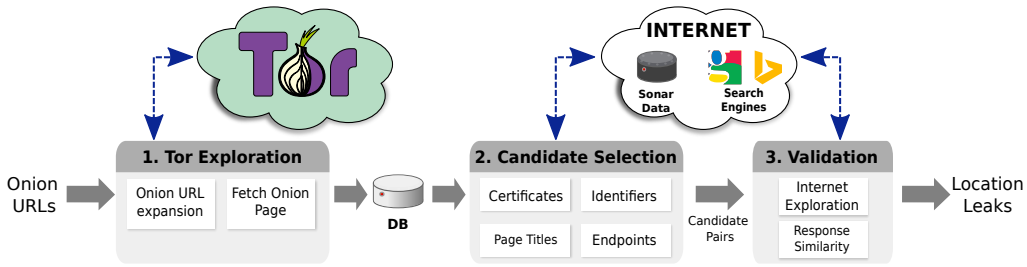

Figure 3.1: Approach overview.

3.2.3 Approach Overview

Figure 3.1 summarizes our approach. It comprises three main steps: *exploration*, *candidate selection*, and *validation*. Exploration takes as input a set of initial onion URLs (Section 3.3.1), creates an extended set of onion URLs and visits them through HTTP and HTTPS to collect the content of the hidden service and its certificate chain (Section 3.3.2). All the information of the exploration is stored in a central database.

Candidate selection takes as input the information in the database and outputs a list of candidate pairs $\langle onion\_address,\ Internet\_endpoint \rangle$, where the Internet endpoint is a DNS domain or IP address considered a candidate to be hosting the hidden server of the onion address (Section 3.3.3). To select candidates, it examines the endpoints (i.e., URLs, email domains, IP addresses) and unique strings (i.e., Google Analytics ID, Google AdSense ID, Bitcoin wallets, page titles) in the onion pages collected from the hidden service, as well as its HTTPS certificate. It searches the unique strings on Internet search engines to find if they are embedded in the content of Internet sites and queries certificate stores (i.e., Sonar [64]) to find Internet sites using the same certificates as the hidden service.

Validation takes as input the candidate pairs and verifies if a candidate Internet endpoint indeed hosts the hidden service (Section 3.3.4). It visits the Internet endpoints to collect their content and certificates. Then, it compares pairs of HTTP responses: one from the candidate Internet endpoint and the other from the hidden service. If it finds a pair with similar content and served from a similar Web server it outputs a location leak.

3.3 Approach

3.3.1 Collecting Onion URLs

CARONTE takes as input a list of onion URLs corresponding to a hidden service to deanonymize. These may be provided by an administrator that wants to check its

hidden service or come from external means. However, for evaluating CARONTE we need to collect onion URLs for a large number of hidden services.

Obtaining onion URLs is a challenging process because there is no centralized repository and valid onion addresses are difficult to predict being a SHA1 hash of the hidden service's key truncated to 80-bit. Owners of hidden services obtain their onion address during installation and are in charge of advertising their onion URLs. Given the nature of some hidden services, their onion URLs may not be publicly advertised, but only disclosed in private forums. In addition, many hidden services do not link to other hidden services, which limits crawling. Furthermore, onion addresses exhibit high churn, as the same hidden service may change onion addresses over time. In 2013, Biryukov et al. [13] presented a technique to list all onion addresses leveraging a flaw in Tor. However, that flaw was fixed after version 0.2.4.10-alpha [12].

As a consequence of these challenges, many sites exist that list onion URLs of hidden services with a description of their content such as "Deepweb links" [23] and "The Hidden Wiki" [111]. There are also specialized search engines for hidden services, typically accessible both from the Internet and through a hidden service [1, 29]. In addition, many onion addresses can be found by querying Web search engines such as Google and Bing.

We have developed scripts that periodically visit the above resources, scrape their links, perform searches on common terms, and feed the identified onion URLs to CARONTE. In addition, as CARONTE explores the hidden services, it adds any new onion URLs found in their content to the list, so that they are visited in the next round of exploration. Overall, CARONTE explores 15 K onion URLs on 6 K onion addresses. On February 2013, Biryukov et al. obtained a complete listing of hidden services and scanned them for open ports [12]. They found 3,741 HTTP hidden services listening on 80/tcp, while CARONTE finds 1,965. Assuming the number of HTTP hidden services has remained relatively constant since then, our sample would cover 52% of them, with popular hidden services being better represented.

3.3.2 Exploring Hidden Services

Given the initial list of onion URLs, CARONTE first creates an extended set of onion URLs that includes for each onion address: the root page, all resources for that onion address in the initial set, and one random resource. The random resource is added to trigger a not found error page, which may leak configuration information specific to the hidden server.

The exploration visits each onion URL in the extended set through Tor eight times: using HTTP and HTTPS, two Host header values (i.e., the onion address and a random onion address), and requesting the resource identified during the collection and a random one. The intuition behind using a random onion address is that the hidden service's Web server may also host other sites. If the hidden

Method	Description
Endpoints	Extract IP addresses, Internet domains in email addresses, and Internet domains in URLs embedded in onion pages.
Identifiers	Use search engines to locate Internet domains embedding Google Analytics, Google AdSense, and Bitcoin wallet identifiers found in onion pages.
Titles	Use search engines to locate Internet domains with similar title as onion page.
Certificates	(1) Extract DNS domains and IP addresses from leaf certificate of hidden service. (2) Search leaf certificate of hidden service in Sonar [64] to obtain IP addresses where observed. (3) Search public key of hidden service certificate in Sonar to obtain IP addresses where observed. (4) Search Sonar for certificates on the Internet with onion addresses and the IP addresses where observed.

Table 3.1: Description of candidate endpoints.

service is not the default site, the Web server may return the default site to the random onion address, and that site may leak some candidate Internet endpoints.

For each request, the exploration collects the response, pre-processes it, and stores both request and response in a database. We limit CARONTE to download only textual and HTML responses (i.e., onion pages) to avoid storing copyrighted software and offensive multimedia content (e.g., explicit videos and pictures). This restriction means that CARONTE cannot identify leaks in other content types, e.g., EXIF data in images, although we do not expect they contain many candidate endpoints. For HTTPS, it also stores the certificate chain provided by the hidden server. The exploration is multi-process and runs from a single host; furthermore this step is rerun over time as new onion URLs are collected.

3.3.3 Identifying Candidate Endpoints

After the exploration finishes one round, the next step is to extract for each onion address a list of candidate Internet endpoints, i.e., DNS domains and IP addresses, which may point to the hidden server. For this, CARONTE examines the endpoints (Section 3.3.3) and unique strings (Section 3.3.3) contained in the *onion pages* collected through Tor, as well as the HTTPS certificate of the hidden server (Section 3.3.3). Table 3.1 summarizes the methods used to obtain candidate endpoints that are detailed in this section.

The input to this step is the information from the exploration stored in the database. This step does not require interacting with the hidden service, but it interacts with some public Internet Web services. The output is a list of candidate pairs ⟨onion_address, Internet_endpoint⟩ where each pair indicates

that the hidden service at the onion address could be hosted at the Internet endpoint. The same onion address may appear in multiple candidate pairs with different Internet endpoints.

Internet Endpoints in Onion Pages

Internet endpoints contained in the onion pages of a hidden service are good candidates for the hidden server. While we expect hidden service administrators to be careful about Internet URLs they link to, there exist several cases in which such leaks can happen, e.g., links or email addresses pointing to other sites of unrelated content but hosted on the same Web server, IP addresses and domains leaked in error pages, and endpoints in comments not visible and possibly forgotten.

For each onion page in the database, CARONTE applies regular expressions to extract URLs, email addresses, and IP addresses in the page. If an extracted URL contains an onion address, it is discarded since we are interested in Internet endpoints, but the onion URL is added to the database if previously unknown. If the URL contains a DNS domain, the domain is checked against the Alexa list of one million most popular domains. Alexa domains are discarded since very popular domains do not typically have a hidden service or publicly advertise it when they do (e.g., Facebook). Non-Alexa domains are added to the list of candidate Internet endpoints for the onion address of the page examined. For email addresses, if the email domain is in a list of popular email providers, it is discarded, otherwise it is a candidate.

IP addresses in the onion page are added as candidates, except when the onion page contains more than 5 IP addresses. In that case, all IP addresses in the page are discarded to prevent large directories of IP addresses unrelated to the hidden server (e.g., lists of Tor relay nodes) to be added as candidates.

Unique Strings in Onion Pages

A general technique to identify candidate Internet endpoints is to first extract some distinctive string from the content of the onion pages of the hidden service and then look up those strings in Internet search engines. Search engines will return Internet sites where they have observed those strings, and their DNS domains can be used as candidate Internet endpoints. This technique can be applied to any unique string that appears in the content of a hidden service. The more unique the string is, i.e., the less it appears in *unrelated* hidden and Internet services, the better the candidates produced. In any case, candidate pairs from non-unique strings are removed during validation. So far, we have added support for two classes of such unique strings: identifiers and page titles. We detail them below and leave as future work applying this technique to other classes of unique strings in the onion pages.

Identifiers. Onion pages may contain identifiers unique to the owners of the hidden service. CARONTE queries dedicated search engines to find Internet sites that also contain those identifiers. Such Internet sites likely belong to the owners of the hidden service and are good candidates for being hosted on the same Web server as the hidden server. Even when they are not, such leaks are dangerous, as those other Internet sites can be monitored by an adversary. If the owners of the hidden service connect to those other Internet sites they own (e.g., to configure them or change their content) without precautions (e.g., without a VPN) their identity could be revealed.

For each onion page, CARONTE applies regular expressions to extract Google Analytics and Google AdSense identifiers, as well as Bitcoin wallets. Google Analytics identifiers are unique to a Google Analytics account and are often embedded in pages to collect statistics. Google AdSense identifiers are less frequently included in pages but are unique for a publisher's account. Bitcoin wallets are often added to ask for donations. An onion page could add the Bitcoin wallet of some other user to encourage donations to that user's site and could even spoof the Google Analytics identifier of an unrelated account. This is not an issue, as unrelated candidates will be discarded during validation.

For each unique identifier extracted from at least one onion page, CARONTE queries dedicated identifier search engines that index Internet pages where identifiers have been observed (e.g., SameID [94]). DNS domains where the identifier has been observed are added as candidates for the onion addresses of the onion pages from where the identifier was extracted.

Titles. Page titles are often specific to the content of a page. Thus, they can also be used as distinctive strings to be looked up in Internet search engines. To reduce the number of queries to search engines, CARONTE first removes any title served by more than 10 hidden services, which removes generic titles such as those of default error pages. Each unique title is then queried on a search engine. DNS domains of the top 10 Internet sites where the title has been observed are added as candidates for the onion address of the onion page with that title.

HTTPS Certificates

CARONTE uses four methods to select candidate pairs from the collected leaf certificate of each HTTPS hidden service. The first method is to extract from the hidden service's certificate the Subject's Common Name (CN) and the Subject Alternative Name (SAN) extension. DNS domains and IP addresses in those fields are added as candidates for any onion address that provided that certificate. The intuition is that Web servers hosting multiple sites often reuse the same leaf certificate for all sites. While it is possible to configure separate certificate chains for each site this relies on the client sending the SNI header. Thus, hidden services' certificates may contain the domains or IP addresses of other Internet

sites hosted on the same Web server.

Even if the certificate of a hidden service does not contain any Internet endpoints, a leak may happen if that certificate (or its public key) is also used by an Internet site on the same Web server. The other 3 methods leverage recently created certificate repositories that store information about certificates that have been observed in Internet traffic (passively, by crawling, or by scanning the Internet). These include Rapid7's Sonar project [64], ICSI's Certificate Notary [56], and Google's Certificate Transparency [45]. ICSI's notary does not provide the IP addresses that served a certificate and Certificate Transparency only allows querying for extended validation (EV) certificates. CARONTE uses Sonar, which contains 35 M unique certificates obtained by periodically scanning the Internet.

The second method computes the SHA1 hash of the DER format of the hidden service's certificate. Then, it uses the hash to search in Sonar whether this certificate has been served by any Internet site. If so, the IP addresses from where the certificate was served are added as candidates for the onion addresses from where CARONTE collected the certificate. Interestingly, some researchers have recently hypothesized that the Silk Road takedown could have been done by correlating the Silk Road's HTTPS certificate with an Internet HTTPS scan as we do [46].

The third method extracts the public key of the hidden service's certificate and searches in Sonar for certificates served by Internet sites that use the same public key. Public keys are left unchanged in many certificate replacements [83]. Thus, it could happen that Sonar observed an older certificate with the same public key in the same IP address still hosting the hidden service.

The fourth method searches in Sonar for any certificate whose Subject's CN or SAN extension contains an onion address. Intuitively, a certificate that contains an onion address but is observed on the Internet likely corresponds to a Web server that hosts both hidden services and Internet sites. This is a wide search for any certificate with an onion address, even if the certificate was not seen during exploration. This method is unlikely to deanonymize a specific target hidden service, but is included for completeness of the measurement study. The onion address and each IP address that served the certificate form a candidate pair. If the onion address was previously unknown, it is queued for exploration.

3.3.4 Validation

Given the list of candidate pairs ⟨onion_address, Internet_endpoint⟩ validation first resolves the candidate domain endpoints to candidate IP addresses. For each candidate pair, it sends eight HTTP requests to the candidate IP address through the Internet. It fetches both the root page and a random resource using HTTP and HTTPS. In addition, it uses two different values of the Host header: the Internet endpoint (domain or IP) and the onion address. The responses and certificate chains are stored in the database. Then, CARONTE computes the

distance between pairs of responses, where one response comes from the Internet endpoint and the other was received through Tor from the hidden service. We detail this distance in Section 3.3.4. If it finds a pair of similar, non-generic, responses the leak is confirmed.

If the candidate endpoint corresponds to a site on the same Web server as the hidden service, that site can serve similar content as the hidden service or a completely unrelated site. If the candidate site is similar, when providing the candidate endpoint in the Host header, the similarity with the hidden service content will manifest. If the sites are different, when provided with the onion address in the Host header the Web server will serve back the hidden service content through the Internet, and the similarity will manifest. The Web server may be configured to serve the hidden service only through Tor, which prevents validation through the Internet. However, validation can also contact the Internet candidate endpoint through Tor, which would bypass this protection. Validation could also be performed using the clock-skew fingerprinting technique from Murdoch [80]. We leave these as future work.

The rest of this section describes the HTTP response similarity metric (Section 3.3.4), classifying leaks as intentional or unintentional (Section 3.3.4), and clustering confirmed leaks on the same servers (Section 3.3.4).

HTTP Response Similarity

Our HTTP response similarity metric takes as input two HTTP responses (one from the hidden service, the other from the candidate Internet endpoint) and outputs one if both responses have similar content and come from a similar server, zero otherwise. When the output is one, the candidate pair is flagged as a location leak.

One challenge is that the similarity should ignore generic pages, e.g., "It works" pages and default error pages, which do not indicate a leak even if observed in both responses. Another challenge is that the similarity needs to handle dynamic content, which may make two equivalent responses be different.

CARONTE uses a simple, yet effective, blacklisting mechanism to identify generic pages. To populate the blacklist, it hashes the body of every page in the database. Hashes that were retrieved from at least 5 different DNS domains or were associated to at least 5 status codes are added to the blacklist. When computing similarity, if any of the two responses has a body whose hash is in the blacklist, the similarity is considered zero and no leak is flagged.

The similarity metric uses 7 features. For each response, it computes the hash of the body and extracts the values of the E-Tag, Last-Modified, Content-Length, Server, and X-Powered-By headers. In addition, it computes the similarity of the two HTML documents using an off-the-shelf package [54], which compares the HTML tag structure of the documents, but not the tag contents. The HTML similarity is high even if both documents contain different dynamic content, as

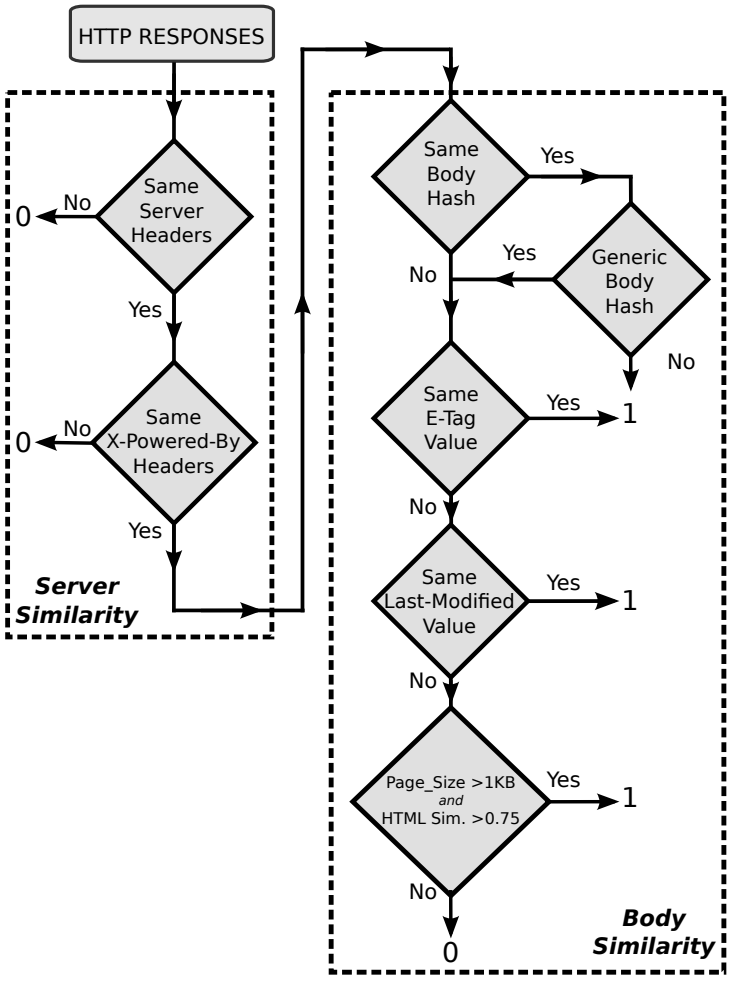

Figure 3.2: HTTP response similarity algorithm

long as the structure of the HTML has not changed significantly.

The similarity metric computation is summarized in Figure 3.2. First, it determines if the content is served from similar servers. For this, it checks whether the Server and X-Powered-By header values are identical. If either the Server or X-Powered-By headers are missing in both responses it considers them identical. If those headers are not identical, it considers the responses different (i.e., outputs zero). Then, it determines that the body of both pages are similar if: they have the same hash and the hash is not in the blacklist, or if they have the same E-Tag value, or the same Last-Modified value, or if both pages are larger than 1 KB and their HTML similarity is greater than 0.75. This threshold allows for small changes in the HTML structure of the pages because the two pages may have been collected at different points in time.

Determining Leak Intention

Next, our approach determines if the leaks are unintentional or not. Clearly we cannot know for sure whether the leak was unintentional, but there exist some indications that the hidden service owners may not be trying to protect their service's location. CARONTE uses 3 automatic checks to classify a leak as intentional.

First, it compares the onion address and the Internet endpoint. If their longest common substring is larger or equal to 4 characters, they are considered similar. Since the onion address comes from a truncated hash, an onion address with a common substring with an Internet domain indicates brute-forcing on the onion address generation (e.g. `facebookcorewwwi.onion` and `www.facebook.com`).

Second, if the Internet site contains the onion address of the hidden service, the leak is also intentional. For example, the Internet site could advertise: "Hidden service available at `niazgxevgzlrbpvq.onion`". This check is only applied to connections where we do not spoof the Host header to prevent errors from servers that echo back the domain provided.

Third, it compares the Internet endpoint with the title of the onion page where the leak was validated. If the title of the page retrieved from the hidden service is embedding the Internet endpoint, likely the owner of the service is intentionally reminding users that the hidden service is also reachable through the Internet. This check is not applied to connections where CARONTE spoofs the Host header.

If none of these 3 conditions is satisfied, CARONTE considers the leak unintentional.

Clustering Leaks

A leak comprises of an onion address and an Internet endpoint (DNS domain or IP address). Multiple leaks may correspond to the same hidden service, e.g., a hidden service may change onion address over time or multiple DNS domains may match an onion address. To simplify the analysis of leaks, CARONTE clusters them based on their onion address, endpoint, and IP addresses.

First, for each leak with a domain endpoint, CARONTE resolves the domain and outputs a tuple ⟨onion_addr, endpoint, IP⟩ for each IP address the domain resolves to. A similar tuple is created for IP endpoints. The clustering uses a distance function that given two tuples returns zero if the onion address, the IP address or the effective second-level domain (ESLD) are the same, otherwise it returns one. The ESLD of a domain is the SLD unless the SLD enables third parties to obtain a subdomain, in which case it is the 3LD. For example, for `www.google.com` the ESLD is `google.com`, and for `books.amazon.co.uk` it is `amazon.co.uk`. To obtain a domain's ESLD CARONTE uses Mozilla's Public Suffix List [78].

The clustering starts with zero clusters and iterates on the list of tuples. For each tuple, if the distance is zero to any tuple already in a cluster it adds the

	All	Live	HTTP	HTTPS
Onion URLs	31,849	4,794	4,617	177
Onion addr.	6,426	1,974	1,965	79

Table 3.2: Onion URLs and addresses in our collection (all), those returning content (live), and their split by protocol.

tuple to the cluster. If the distance is zero to tuples in different clusters, it merges those clusters and adds the tuple to the merged cluster. Otherwise, it creates a new cluster for the tuple.

3.4 Evaluation

This section details the evaluation of CARONTE: the datasets used (Section 4.3), the selection of candidate pairs (Section 3.4.2) and the validation of the candidate pairs (Section 3.4.3).

3.4.1 Datasets

Table 3.2 summarizes the collection and exploration of onion URLs. It shows the number of onion URLs and addresses initially collected from search engines and indices (*All*), the total number of onion URLs that returned some content (*Live*), and the split by protocol. Overall, CARONTE explored 31,849 onion URLs from 6,426 onion address. Only 31% of the onion addresses were alive, which may be due to non-HTTP(S) services, short-lived hidden services, and hidden services that change onion address over time. Only 4% of the live onion addresses offered HTTPS, which may be due to the communication with hidden services being encrypted in all hops and the difficulty of having a CA sign a certificate for an onion address. The exploration took place in 4 rounds, each round lasting three days on average.

Sonar data. The Sonar project [64] offers data from 68 Internet-wide scans performed between October 2013 and February 2015. Each scan has 3 files. The *certs* file (average size of 448MB per scan) has one row for each certificate with its SHA1 hash in DER format and the certificate in base64 encoding. Each row in the *names* file (30 MB) contains the certificate hash and the Subject's CN. The *hosts* file (2 GB) contains in each row the certificate hash and an IP address from where the certificate was collected. Overall, the Sonar data comprises 205 GB and contains 35 M distinct certificates.

Type	Endpoints	Onion Addr.	Cand.
URLs	6,207 (11,207)	916 (2,301)	4,372
Domains	182 (231)	172 (269)	249
IPs	73 (22,182)	75 (89)	102
TOTAL			4,704

Table 3.3: Endpoints extracted from onion pages.

3.4.2 Candidate Pairs

This section describes the candidate pairs extracted from the endpoints, identifiers, and certificates.

Endpoints. Table 3.3 summarizes the endpoints extracted from the onion pages collected during exploration. For each type of endpoint, it shows the number of distinct endpoints after filtering (before filtering in brackets), the number of onion addresses with at least one endpoint after filtering (before filtering in brackets), and the number of candidate pairs obtained from that type of endpoint. The numbers show that the filtering of non-Alexa domains and pages with more than 5 IP addresses removes 81% of the endpoints, most of those being IP addresses from Tor relay status pages. After filtering, the median contribution of each onion address is: 2 URLs, 1 domain from email addresses and 1 IP address. CARONTE produces 4,704 candidate pairs from endpoints: 93% from URLs, 5% from email domains, and 2% from IP addresses.

Identifiers. Table 3.4 summarizes the identifiers extracted from the onion pages collected during exploration. For each type of identifier, it shows the number of distinct IDs, the number of onion addresses and URLs containing at least one identifier of that type, and the number of candidate pairs obtained from these identifiers by searching on Internet search engines. CARONTE extracted 58 unique identifiers: 24 Google Analytics IDs, 3 Google AdSense IDs, and 31 Bitcoin wallets. Only 66 hidden services (3.3%) contained an identifier, indicating that most administrators are careful to avoid them. After querying the 58 identifiers on search engines, CARONTE found candidate Internet endpoints for 32 (64%), for a total of 192 candidate pairs.

Titles. After filtering titles in onion pages that appear in more than 10 hidden services CARONTE identifies 583 distinct titles. Looking up these titles on a search engine returns hits for 183 titles. From those hits, CARONTE produces 200 candidate pairs.

Certificates. Of the 79 onion addresses with port 443/tcp open, 50 provided a certificate chain. CARONTE uses 4 methods to obtain candidate pairs from

		Onion		Cand.
Identifier	**IDs**	**Addr.**	**URLs**	**Pairs**
Google Analytics	24	33	52	146
Google AdSense	3	3	3	24
Bitcoin	31	31	36	22
TOTAL	*58*	*66*	*90*	*192*

Table 3.4: Identifiers extracted from the onion pages.

HTTPS certificates, which are summarized in Table 3.5. The table shows the number of candidate pairs, and the number of distinct onion addresses, IP addresses, and DNS domains in those pairs. CARONTE produces a total of 366 candidate pairs from certificates. Next, we detail the results for each method.

First, CARONTE extracts DNS domains and IP addresses from the Subject's CN and SAN extension from each leaf certificate collected during exploration. Of the 50 leaf certificates, 11 (22%) contain only onion addresses, 3 (6%) contain both Internet endpoints and onion addresses, 26 (52%) contain only Internet endpoints, and 10 (20%) contain no Internet endpoint or onion address. The 3 certificates with both onion and Internet endpoints are likely doing this on purpose. However, 52% of the HTTPS hidden services are leaking endpoints in their certificate. This method produces 81 candidate pairs for 29 onion addresses (2.8 candidate endpoints per onion address).

Second, CARONTE searches the hash of the 50 leaf certificates in Sonar. It finds 30 of them, so at least 60% of the HTTPS hidden services are using the same certificate that an Internet Web server does. Of those 30, 26 were identified by the previous method. This method identifies 4 certificates that were not leaking an Internet endpoint but can still be found on Internet Web servers. Of those 4, 3 contain no Internet endpoints or onion addresses and the other one an onion address only. For each of the 30 certificates, Sonar provides one or more IP addresses from where the certificates were collected for a total of 188 candidate pairs.

Third, CARONTE searches the public keys of the 50 leaf certificates in Sonar. This method finds 9 new certificates that share their public key with one of the 50 leaf certificates. Those 9 certificates are observed being distributed from 20 IPs, of which 5 are not known from the previous methods.

Fourth, CARONTE searches Sonar for certificates that contain an onion address in their Subject CN or SAN extension. It finds 30 previously unknown certificates (i.e., not observed during exploration), each with one onion address. Some of those are additional Facebook certificates, which are excluded because they generate many useless candidate pairs. The IP addresses from where the remaining certificates were observed produce 97 candidate pairs.

	Candidates			
Method	**Pairs**	**Onion Addr.**	**IPs**	**Dom.**
Method 1	81	29	0	81
Method 2	188	30	188	0
Method 3	20	9	20	0
Method 4	97	30	37	60
TOTAL	*366*	*63*	*225*	*141*

Table 3.5: Candidates pairs from certificates.

Total. The left side of Table 3.6 summarizes the candidate pairs per method. In summary, CARONTE produced 5,462 candidate pairs: 86% from endpoints, 3% from identifiers, 4% from titles, and 7% from certificates.

3.4.3 Validation

Of the 5,462 candidate pairs, 303 (5.8%) successfully validated. Of those 303 pairs, 88 had an IP address as Internet endpoint and 215 a domain name. Those 303 pairs contained 100 unique onion addresses, 87 IP addresses, and 163 domains. One of the onion addresses offered two different hidden services, one through HTTP and the other through HTTPS. Thus, 101 hidden services were successfully deanonymized by CARONTE. Next, we classify leaks as intentional or not.

Intentional leaks. CARONTE automatically labels as intentional those leaks that match any of the 3 rules described in Section 3.3.4. Overall, 49% deanonymized hidden services are considered to be leaking their location intentionally. For these, the hidden service has an Internet site counterpart that serves similar content and runs on the same Web server, and there are references from the onion site to the Internet site and/or viceversa, e.g., the onion address appearing in the Internet site or the title of the onion site being similar to the domain of the Internet site. This result indicates that almost half of the hidden services CARONTE deanonymized may not be trying to protect the hidden server's location. This may be due to a selection bias in our collection towards the most advertised hidden services. One benefit to these Internet sites of having a hidden service is that users of the hidden service have their communication encrypted at all hops from the client to the hidden service. Furthermore, since the onion address is derived from the public key through a hash, the public key is self-authenticating, which helps preventing man-in-the-middle attacks.

Unintentional leaks. We manually analyze 15 of the unintentional leaks. They correspond to 4 cases. First, 5 hidden services run on a Web server with multiple virtual hosts, where the hidden service is not the default virtual host.

Method	Candidates		Deanonymizations	
	Pairs	**Onions**	**All**	**Unintentional**
Endpoints	4,704	793	67	32
Identifiers	192	66	12	2
Titles	200	157	44	20
Certificates	366	63	30	18
TOTAL	5,462	841	101	51

Table 3.6: Summary of location leaks.

When connecting to the hidden service providing a random domain in the Host header the Web server provides the default site, different from the hidden service, which contains the candidate Internet endpoints.

Second, 8 hidden services are deanonymized through certificate leaks, which provide a candidate Internet endpoint, even if the content of the hidden service did not provide any candidate. Of these, 4 are certificates that include the domain or IP address of the hidden server; the other 4 are certificates with an onion address observed by Sonar but not in our exploration.

Another hidden service is deanonymized because it contains an email address to a DNS domain with an Internet site also hosted in the same Web server. While the Internet site serves content unrelated to the hidden server, when querying the Internet site with the Host header being the onion address, the hidden service content is returned through the Internet.

The last deanonymization happens with two hidden services running on the same onion address through HTTP and HTTPS, respectively. One has an intentional leak to an Internet site offering the same content through both HTTP and HTTPS. When requesting the Internet site with the Host header being the onion address the two hidden services are observed, one on each port.

Location leak types. Table 3.6 summarizes the candidate pairs and deanonymizations per method. For the candidates it shows the total number of pairs and the distinct onion addresses in those pairs. The deanonymizations show the total onion addresses deanonymized by each method and how many of those are classified as unintentional. When considering all 101 deanonymized hidden services, URL endpoints are the most effective content leak being present in 67 deanonymizations, followed by titles (44), certificates (30), and identifiers (12). The identifiers correspond to Google Analytics (8) and Bitcoin wallets (4). Note that some deanonymizations contain several types of leaks. When considering only unintentional leaks, the most effective location leaks are endpoints (32), titles (20), and certificates (18). These numbers indicate that certificate leaks are best to identify unintentional leaks (60%) and identifiers worst (17%), which may indicate that administrators are already careful about them.

Clustering. Grouping the 101 hidden services by shared domains and IP addresses leaves 80 clusters. Of those, 12 contain multiple onion addresses. Three of those 12 correspond to hidden services that offer the same content through different onion addresses. The other 9 clusters with multiple onion addresses correspond to servers that are being used for hosting multiple hidden services.

Tor relay hosting. Of the 101 hidden services deanonymized, 21 are hosted on 13 Tor relays. Of those, 18 are considered intentional leaks and 3 unintentional, a smaller fraction of unintentional leaks compared with non-relay servers. This matches reports by the Tor Project that several Tor relays were taken down during operation Onymous [109]. In some cases multiple hidden services are hosted at the same Tor relay, indicating that those hidden services belong to the owner of the Tor relay or that the Tor relay provides hidden service hosting services. Note that the Tor Project discourages hosting hidden services on Tor relays as Tor relay runtime is known and can be correlated with the uptime of a hidden service [114]. Furthermore, CARONTE could be configured to consider the IP addresses of all Tor relays as candidates for all hidden services. We leave this as future work as our goal is to demonstrate open-world deanonymization and also to avoid impacting Tor relays.

Redirections. Fourteen hidden services redirect to an Internet site counterpart at some point of the study. Surprisingly, 9 of these hidden services redirect to the HTTP version of the Internet site. This is problematic for their users as they expect all hops of the communication to be encrypted when accessing a hidden service, but the last hop will not be. For these sites, the benefit of having a hidden service is not clear.

3.4.4 Performance

CARONTE currently runs from one off-the-shelf workstation. Once preprocessing finishes (e.g., parsing/indexing Sonar records) analyzing a hidden service takes close to one minute. Most time is spent in network connections. To scale CARONTE we could use multiple backend servers. Once a hidden service to be deanonymized is submitted a link could be generated where results would be available after the analysis completed.

3.5 Defenses

This section summarizes best practices that administrators of hidden services should take to eliminate location leaks. Some of these best practices are mentioned at the Tor Project's how-to configure a hidden service [114].

Use a dedicated Web server. CARONTE can validate candidate domains when the same Web server is used to host both a hidden server and an Internet site. The Tor Project recommends installing the hidden service in a separate Web server, but we believe this is a *must* and we recommend changing the wording to emphasize this. Administrators should specifically avoid reusing an existing Web server that already hosts an Internet site by simply creating a new virtual host. It is also a good idea to host the hidden service on a dedicated machine.

Bind the web server to localhost. Another best practice is binding the hidden service's Web server only to localhost, so that HTTP requests from Tor (running as a SOCKS proxy on localhost) are successfully answered, but HTTP requests from the Internet receive an HTTP Forbidden error response. We emphasize that this protection is required *in addition* to running the hidden server on a dedicated Web server. More specifically, if the hidden server is hosted on the same Web server as another Internet site, even if the Web server is bound to localhost, it is still possible to perform validation by contacting the Internet site over Tor, rather than through the Internet. To prevent configuration errors we recommend the Tor Project to add detailed instructions on how to configure a Web server in this manner, at least for the most common open source Web servers.

A firewall can also block non-Tor connections to the hidden server. Network firewalls can block connections that do not come from a Tor relay. However, the list of Tor relays needs to be continuously updated. Host firewalls can only allow connections from localhost. Similar to binding the Web server to localhost, the hidden service should still be run on a dedicated Web server. The advantage is that Internet connections fail without generating an error message.

Another principle could be that if the server does not know its IP address it cannot leak it. Thus, the hidden service could be run in a NATed VM without a public IP address. However, other leaks (e.g., domain names in the content) could still happen.

Site auditing. Administrators should carefully audit their site's contents for leaks before making the site available as a hidden service. A general rule is that given the many possible sources of location leaks, the smaller the site, the easier the auditing and the smaller the probability of location leaks. During the audit, administrators could use CARONTE to identify validated location leaks, as well as to examine the candidate pairs CARONTE could not validate. Also, all external links should be made to point to onion addresses rather than Internet domains or IP addresses. Since not all Internet sites have a corresponding onion address, they should also audit all remaining IP addresses and Internet domains so that they do not point to the public IP address of the hidden server. They should also remove links to Internet domains or IP addresses owned by them, even when hosted in other servers, as tracking accesses to those other servers may reveal the

identity of the administrators. Furthermore, identifiers such as email addresses, Google Analytics, Google AdSense, and Bitcoin addresses should be removed or at least not be reused in any other sites.

Certificates. Certificates with onion addresses in the CN or SAN fields should not include DNS domains or IP addresses, and should exclusively be used on Web servers that only run a hidden service bound to localhost. In addition, a hidden service should never reuse the certificate chain of another Internet site from the owner. Nowadays, the vast majority of hidden services use self-signed certificates. Any entity willing to obtain a proper TLS certificate, is required to pass the Extended Validation test, which includes verification entity's identity by the CA. This does not fit to hidden services, which by definition are anonymous excluding some exceptions (e.g., Facebook's hidden service seems to have a valid certificate chain [92]). Thus, certificate chains do not validate and will generate warnings, but man-in-the-middle attacks are limited by the fact that public keys of hidden services are self-authenticating.

Avoid Tor relays. Hidden services that want to hide their location should not be hosted on Tor relays, as this enables attackers to perform closed-world validation using the IP addresses of all Tor relays as candidates for any hidden service to be deanonymized. Also, Tor relay uptime is public and can be correlated with hidden service uptime. Similarly, running a hidden service on a machine that is not always on enables uptime correlation attacks.

Other attacks. Some other recommendations are designed to protect against network-level adversaries or code vulnerabilities. For example, not running a hidden service on a Tor relay or on a machine that is not always on, as host uptime can be correlated with traffic to leak the server's identity. Or, running the Web server of the hidden service in a sandbox to limit the impact of software vulnerabilities. Applications as nitko [18], DirBuster [97] and theHarverster [19] can be used to automate the process of finding possible leaks in the contents accessible through the web application. Other tools such as w3af [4] and sqlmap [11] can perform even more aggressive tests, attempting to find and exploit exploiting web application vulnerabilities.

While all the recommendations mentioned above do not make the hidden service resistant to any kind of leaks, they represent good practices that every owner of a hidden service should take care. Following these guidelines it is possible to harden the service and reduce sensibly risks of unintentional leaks about the physical location of the server or owner's identity.

3.6 Ethical Considerations

While there exist tools like Shadow [57] to simulate attacks against Tor, the application layer leaks studied in this paper cannot be easily simulated because they are caused by erroneous configurations by administrators of hidden services. Similar to Biryukov et al. [13] we deem experiments on the live Tor network worthwhile and necessary to enhance the scientific understanding of hidden services, as long as they do not cause degradation of the network and hidden services.

Our approach does not add any malicious Tor relay and gives us no access to the plaintext of traffic of any user. We purposefully avoid deanonymization techniques based on exploiting software vulnerabilities. However, our data is sensitive because it contains location information on some hidden services that may want to protect their location. This work has been approved by our institution's ethics review board, which has mandated that due to its sensitive nature the data must be protected with diligence, must not be disclosed to third parties, and must be deleted when the paper is accepted for publication. Furthermore, in this paper we do not disclose any deanonymized hidden service but only provide aggregate data or fake examples to illustrate important steps and findings.

During the exploration of hidden services, to prevent downloading copyrighted material and offensive content (e.g., pornography), we limit CARONTE to collecting textual and HTML content, ignoring other content such as images, videos, or documents.

All the design choices mentioned above strictly comply with the list of guidelines that Tor Project recommend researchers to follow in order to conduct responsible research [105]. Not just that we access exclusively publicly available information and datasets, but we also have access only to our own traffic and we try to minimize the contents downloaded from hidden services in order to limit network load.

We have sent a copy of this work to the Tor Project to give them an advance notice of our work.

3.7 Related Work

The first generation of Tor's hidden services is described in the original design paper [28], but has since been revised [24]. Øverlier and Syverson [86] first demonstrated techniques to deanonymize hidden services. They show how an adversary could lie about the available bandwidth of a relay it controls to increase the probability of that relay being selected for a hidden service circuit. This adversary can repeatedly connect to a hidden server until traffic correlation indicates that the hidden server built a circuit to one of the adversary's relays. Bauer et al. [10] extended the attack to general purpose circuits. As the result of these attacks, entry guard nodes were added to the Tor hidden services specification [24,130]. Elahi et

al. [34] propose improvements to reduce the guard compromise rate. Biryukov et al. [13] deanonymize hidden services in the presence of guard nodes by combining the bandwidth inflation attack with a technique to phase their relays in and out of the consensus at will without them losing their flags.

These attacks target a specific hidden service, while we show that large-scale deanonymization of many hidden services is possible through content leaks. Also, these attacks rely on vulnerabilities in the Tor specification that can be centrally addressed by the Tor Project, while content leaks need to be fixed by each hidden service. Furthermore, our attacks do not require adding a malicious relay to the Tor network.

Most similar to our approach is the work by Crenshaw on deanonymizing I2P eepSites [22]. Crenshaw sets up an I2P router and uses the IP addresses of the peers known to his router as candidates for hosting the eepSites. In Tor, which is not peer-to-peer based, his approach is similar to considering the list of all Tor relays as candidates for hosting any hidden service. This closed-world approach would enable deanonymizing only 21% of hidden services that are hosted on Tor relays. Our approach in contrast shows how to move from an open-world to a closed-world by extracting candidates from the identifiers and endpoints in the content of hidden services, as well as their HTTPS certificates. In addition, his validation step relies solely on similar Server headers, which can produce a high number of false positives.

A different deanonymization approach uses clock-skew measurements when repeatedly connecting to a hidden service [80, 132]. This attack also assumes a closed-world. We propose techniques to move from a closed-world to an open-world and could also use this technique in our validation.

DoS attacks. Selective denial-of-service attacks on relays can force circuits to be re-built, increasing the probability of end-to-end compromise [15]. Øverlier and Syverson [87] introduce Valet Service nodes to improve the resilience of introduction points against DoS attacks. Jansen et al. [58] deanonymize hidden services through selective denial-of-service of relays' memory that forces the hidden service to choose guard nodes in control of the adversary. The attack requires hours or days to deanonymize a single hidden service and requires the adversary to identify the target's guards.

Forensics. A separate line of work considers the forensics problem of proving that a confiscated machine hosted a hidden service [36, 96]. Those works assume the hidden service logs the requests and place identifiable fingerprints in the log files through crafted queries.

AS-level adversaries. An attacker that is able to observe encrypted traffic from a client to the first relay and from the final relay to the destination can

link the client and destination by correlating traffic patterns [99]. Prior work has studied Tor's vulnerability to adversaries that can establish themselves in that position both at the Internet exchange [82] and the AS [2, 33, 40, 59, 118] levels. Our attacks instead deanonymize hidden services.

Traffic analysis. Prior work proposes traffic analysis attacks on Tor that make probabilistic inferences about relays and clients that are part of a specific circuit. One approach is to congest a relay [38, 81]. Another approach leverages network-level characteristics such as circuit throughput and latency [50, 75]. These attacks target clients and may not fully deanonymize them, while our attacks fully deanonymize the location of hidden servers.

3.8 Conclusion

In this paper we have presented CARONTE, a tool to deanonymize hidden services through location leaks in their content and configuration. CARONTE implements a novel approach to deanonymize hidden services that does not rely on flaws on the Tor protocol and assumes an open-world, i.e., it does not assume a short list of candidate servers is known in advance. Instead, it implements novel techniques to identify candidate servers from the content and configuration of a hidden service, which enable moving from an open-world to a closed-world.

Using CARONTE we perform the first measurement study on the prevalence of location leaks in hidden services. Out of 1,974 live HTTP hidden services, CARONTE successfully deanonymizes the location of 5% of them. Of the deanonymized hidden services 21% are running on Tor relays. The remaining 79% could not be deanonymized in a close-world.

3.9 Acknowledgments

The authors would like to thank the anonymous reviewers for their feedback. This work was performed while Srdjan Matic was a visiting Ph.D. student at the IMDEA Software Institute.

This research was partially supported by the Regional Government of Madrid through the N-GREENS Software-CM project
S2013/ICE-2731 and by the Spanish Government through the StrongSoft Grant TIN2012-39391-C04-01. All opinions, findings and conclusions, or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the sponsors.

4

Dissecting Tor Bridges: a Security Evaluation of Their Private and Public Infrastructures

This chapter reproduces the work "Dissecting Tor Bridges: a Security Evaluation of Their Private and Public Infrastructures" that will appear in NDSS 2017. This paper performs the first systematic study of the public and private Tor bridge infrastructure. This work was realized together with people from the IMDEA Software Institute.

4.1 Introduction

The Tor Network [28] offers protection against censorship [116], surveillance, and traffic monitoring [100, 129], by using encryption and hiding communication patterns by routing traffic through several *onion routers* (ORs). Tor keeps secret the IP addresses of a fraction of the entry ORs, called *bridges*, so that it is not trivial to block traffic destined to the Tor Network. To increase protection, bridges deploy *Pluggable Transports* (PTs) [108], which disguise Tor traffic so that it is difficult to recognize through deep packet inspection [37, 119] or active probing approaches that connect to the bridge mimicking a Tor user [122, 127].

Research related to Tor bridges has focused on designing secure PTs [5, 76, 98, 120, 128], and proposing techniques to discover the IP address of bridges [66, 74]. However, to date there has been no security analysis of the bridge infrastructure as a whole. In this work we set out to perform the first systematic study of the Tor bridge infrastructure from a security point of view. We study both the infrastructure of *public bridges*, i.e., bridges that volunteers provide to be used by any Tor user, and *private bridges*, i.e., bridges for the exclusive use of individuals who know about their existence. While public bridges are known to the Tor Network, and report configuration and usage information to *bridge authorities*, private bridges do not report such data and thus their population and properties remain unknown. As far as we know, this is the first work that reports on Tor's

private bridge infrastructure.

Public Infrastructure Analysis. To study public bridges, we leverage CollecTor [104], a public Tor service that enables access to fine-grained longitudinal data reported by public bridges (among other Tor nodes). The goal of our analysis of the public infrastructure is twofold. First, we aim at understanding whether any of the published data can harm the security of the public bridge infrastructure. We find out that usage statistics in CollecTor can be used to rank bridges by importance, e.g., by the number of clients from a specific country, or the amount of traffic carried for a particularly strong PT, which in turn allows an adversary to evaluate how well her blocking works and to identify targets. Furthermore, we find that the publication of which *OR port* a bridge uses to communicate with bridge authorities can be leveraged to optimize a scan-based search for IP addresses of public bridges and to select specific ports to scan to find a target bridge. Our findings, already reported to the Tor Project, have lead CollecTor maintainers to start sanitizing the OR port data [69].

Second, we aim at measuring security-relevant properties of public bridges. We analyze the population size and its stability, finding that only 45% of public bridges carry user traffic. These bridges are long-lived and stable: their median lifetime is 4 months, and they rarely change IP address. While stability is good to increase bridge usage, it also means an adversary that discovers a bridge can block it for long periods of time without side-effects. We also observe that *default bridges*, whose IP addresses can be obtained from the Tor client configuration, support over 90% of bridge users, essentially defeating the very purpose of bridges. This holds in countries where censorship is known to happen such as Iran or Syria, raising the issue that a censor can at any point, e.g., in response to an event like the recent coup in Turkey, disconnect the vast majority of bridge users in the country. Finally, we analyze PT deployment, finding that 77% of public bridges only offer the easy-to-detect "vanilla Tor" and another 15% offer PTs with conflicting security properties (e.g., with and without active probing protection). The latter enables an adversary to identify the bridge through the weaker PT and disable the stronger PTs by blocking the bridge's IP.

Private Infrastructure Analysis. To study private bridges, not present in CollecTor or other Tor services, we first leverage two known Tor issues [31, 117] to find their IP addresses using Internet-wide scans. This means that our study of the private infrastructure is opportunistic; fixing the Tor issues we leverage may prevent future replication of our measurements. Rather than launching our own scans, which could interfere with the Tor Network, we show how scan search engines [30,72] can be leveraged to launch large-scale discovery of the IP addresses of bridges, with no investment in scanning infrastructure.

We follow an approach to discover bridges that uses scan search engines to find candidate IP addresses that may run an OR; connects to them to confirm their OR

role; and uses CollecTor data to filter out relays and classify discovered bridges as public or private. Without launching any scan, our approach discovers 694 private bridges, and deanonymizes the IP address of 35% of public bridges with clients, and 23% of all active public bridges, in April 2016. Of all discovered bridges, 65% are public and 35% are private. We also propose a novel technique to track known bridges across IP address changes. This technique leverages additional non-Tor services (e.g., SSH) running on bridge hosts, and can be used even if the two Tor issues we leverage are fixed by the Tor Project.

In the process of discovering bridges' IPs, we also uncover 645 *private proxies*, i.e., private IP addresses that forward traffic to a backend bridge or relay, and through which users can also enter the Tor Network. As far as we know, we are first to report on the existence of such private proxies. An important security implication is that discovery of a private bridge or proxy enables an adversary to flag IP addresses connecting to it as members of the owner organization, or the owner itself, and to geographically locate them.

We study the infrastructures built using proxies and bridges using a novel clustering approach to group ORs owned by the same entity based on their configuration and IP addresses. We observe 3 prevalent cluster types: (I) a line of 2 up to 178 proxies on nearby IP addresses all forwarding to the same backend OR; (II) a simplified version, with a single proxy forwarding to one backend OR; and (III) a set of bridges with no proxies, where bridges are either all public or all private. In both Type I and Type II clusters, the backend OR is typically a public bridge or a relay (but rarely a private bridge), and in 77% of these clusters the backend is in the same autonomous system (AS) as the proxies. In other words, cluster owners seem to contribute a public bridge or relay to the Tor Network, but use nearby IP addresses to run private proxies for their exclusive use. However, in general these proxies do not contribute much IP address diversity as they are hosted in the same AS and typically in nearby IP addresses. In 93% of Type III clusters, all bridges are located in the same AS, thus also raising concerns on lack of IP diversity.

4.2 Overview

We now present an overview of the Tor Network, focused on the components more relevant to our work. Then, in Section 4.2.2, we describe open issues in Tor that an adversary can leverage to discover the IPs of hosts running bridges.

4.2.1 The Tor Network

The core element of the Tor Network are *Onion Routers* (ORs), also known as *relays*, which are essentially routers that forward encrypted data. A user that wants to anonymously access an Internet service runs the Tor software on its

client host. This software builds a circuit of connections over three ORs, through which traffic is forwarded between the client host and Internet services. This circuit guarantees that the traffic is encrypted until it exits the circuit and that none of the relays knows both the origin and the destination of the traffic. Some of the ORs act as central authorities known as *directory servers*, storing contact information for all ORs currently part of the Tor Network. Directory servers can be queried by clients to find relays when building circuits.

Each OR is uniquely identified in the Tor Network by its *fingerprint*, which is the 20-byte SHA1 hash of its public key. ORs listen on a dedicated *OR port* for incoming connections using the *vanilla Tor protocol* [27]. The OR port is by default set to 0 [89], i.e., a freshly installed OR will not accept connections. To use the OR as a relay or bridge, the owner needs to explicitly set the OR port in the configuration file to a particular port, or to `auto` to choose a random OR port.

Bridges. Since the IP addresses of all Tor relays can be obtained at any point if time from the directory servers, the Tor Network introduced a new OR type called *bridge*. Bridges are essentially relays that act always as first hop in a circuit, and whose IP addresses are not publicly advertised.

Pluggable Transports. An alternative way to prevent access to the Tor Network is blocking any traffic that looks like Tor communication, regardless of its destination. This is possible due to distinguishing features of "vanilla Tor" that are easy to detect (detailed in Section 4.2.2). After censors started deploying deep packet inspection techniques to detect such features, the Tor Network introduced *Pluggable Transports* (PTs) [108]. A PT is just a wrapper for the Tor protocol that transforms the Tor traffic flowing between clients and bridges. Over time multiple PTs have been proposed and the most recent protocols include features as reply protection, which guarantees that the bridge will allow connections and data transmits, only for users that previously authenticated. Pluggable Transports either imitate popular protocols (e.g., fte [32]), encapsulate Tor traffic using popular protocols like TLS (e.g., meek [42]), or are designed to look like random streams (e.g., obfs3 [88]). PTs may also implement reply protection against active probing (e.g., obfs4, ScrambleSuit), in which case they require users to know a shared secret before replying. A bridge can offer multiple PTs, each running on its own *PT port*.

Bridge Distribution. To use a bridge, Tor clients need to obtain its endpoint information, i.e., the IP address and port where the bridge listens for connections. Additionally, the user may need some extra information (e.g., the secret when using PTs with reply protection). Since it must not be possible for an adversary to find out the IP address of all bridges, their endpoint information needs to

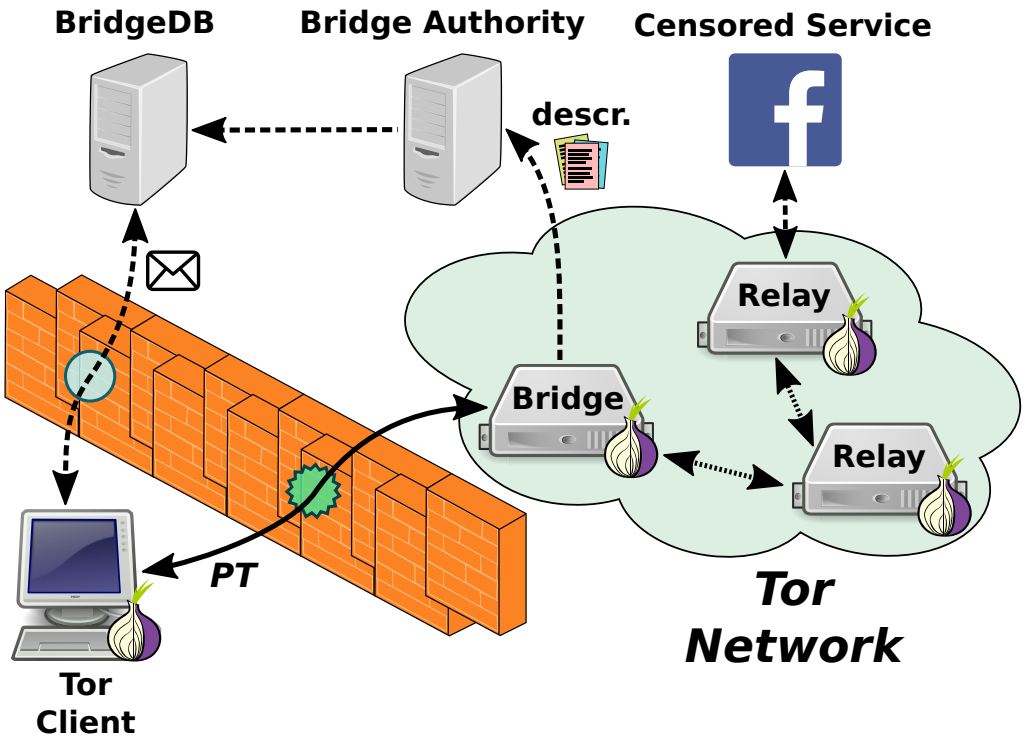

Figure 4.1: Bridge distribution workflow: ❶ upon installation the bridge sends its descriptors with contact information to the Bridge Authority that ❷ assigns the bridge to BridgeDB; ❸ the user requests a bridge from BridgeDB through an uncensored channel such as email; ❹ BridgeDB sends the contact information for the bridge; ❺ the client connects to the bridge using a PT; ❻ and the bridge builds a circuit through the Tor Network. At this point, ❼ the client can communicate to the censored service through Tor.

be carefully distributed to clients. There exist two classes of bridges: *public* and *private*. Public bridges can be used by any Tor client. They upload their endpoint information to Tor's *Bridge Authority* (or bridge directory authority), which maintains a list of available public bridges in the Tor Network. Endpoint information for public bridges is distributed to users by the *BridgeDB* service, which periodically receives it from the Bridge Authority. BridgeDB supports two different distribution channels. Users can visit its website[1] or send it an email request[2]. In both cases, users can specify the type of transport they want and whether they need a bridge that supports IPv6. Figure 4.1 depicts the distribution workflow for a public bridge.

The distribution algorithm adopted by BridgeDB aims at preventing the list-

[1] https://bridges.torproject.org/
[2] bridges@torproject.org

ing of a significant fraction of public bridges [71]: it only distributes a few bridges to each requesting IP address or email account, it restricts distribution to a subset of the bridge pool that changes over time, and it limits email requests to addresses from specific mail providers (Gmail, Yahoo, RiseUp).

To facilitate the use of bridges without having to go through the BridgeDB distribution channels, the Tor software ships with a list of *default bridges* for different transports. The IP addresses of these bridges are trivial to obtain, since they are hard coded in the Tor configuration files – effectively making them relays from the point of view of an adversary. Thus, these bridges can be easily blocked by adversaries. When the Tor Project detects blocking on a default bridge, the bridge is replaced by a new default.

In contrast, private bridges do not share their endpoint information with the Bridge Authority and thus are opaque to the Tor Project maintainers. Since they do not upload their descriptors on the Bridge Authority, they are not advertised to users that request a bridge to BridgeDN. Endpoint information of private bridges is distributed using private channels shared between the operator running the private bridges and the people that use them.

4.2.2 Known Tor Issues

In this Section we describe two known Tor open issues that we leverage to discover IP addresses of bridges.

Vanilla Tor Certificates. The "vanilla Tor" protocol comprises two phases. First, the client and the bridge perform a TLS handshake to agree on a shared key. Then, the two parties exchange Tor messages encrypted with that shared key. In principle, using a TLS handshake should make the "vanilla Tor" traffic look like TLS. In practice, the certificate chain sent by the bridge to the client during the TLS handshake is easily distinguishable, enabling to identify Tor handshakes among all TLS handshakes. In particular, the certificate chain contains a single certificate where the subject and the issuer differ and their common names have an easy to identify pattern: SubjectCN=www.[random].com; IssuerCN=www.[random].net, where [random] are base32-encoded random strings of length between 8 and 20 characters. While the certificate is changed every 2 hours, this pattern is always maintained. This issue is known by the Tor Project since at least October 2012, when a ticket was open to revise the certificates used by Tor [68]. However, the conclusion was that efforts to make vanilla Tor indistinguishable from TLS were superseded by the introduction of Pluggable Transports, and the issue was left as "wontfix" [68]. This decision should be ascribed mainly to the choice to do not change code in core parts of Tor, with the risk of introducing new bugs and security issues. Furthermore, developers believed that PTs were deployed widely enough for being considered the state of the art solution for users needing to bypass censorship.

Open OR port in Bridges. The second known open issue is that bridges always have an OR port open that offers vanilla Tor, even when they do not advertise vanilla Tor as a transport, but only advertise stronger PTs. Thus, bridges offering PTs will open one port per PT plus an additional one for the OR port. This issue is also known since at least November 2012, when a ticket was open for it [7]. In September 2015, the ticket priority was increased as it was considered the next major defense against bridge enumeration. But, it was also stated that the fix may require up to a month of work, as it requires changes to the Bridge Authority and BridgeDB, as well as examining multiple tools that assume bridges have an OR port.

4.3 Public Data Sources

We leverage two types of *publicly available* services as sources of data for analyzing the security of Tor bridges. On the one hand we use data published by the Tor Project through the CollecTor service [104], which provides fine-grained configuration information and usage statistics about individual bridges and relays over time. On the other hand, we use data obtained from scan search engines, which provide information about services offered on machines connected to the Internet.

4.3.1 CollecTor

CollecTor is a service offered by the Tor Network that periodically collects data from Tor relays, public bridges, and other Tor services, and makes it available online [104]. In contrast to other Tor services that provide aggregated information on the whole Tor Network (e.g., Tor Metrics [110]), CollecTor provides information at the finer granularity of individual ORs (bridges or relays). In this paper we only consider data published since July 2012, when CollecTor started to include statistics on Pluggable Transports, until April 30th 2016.

CollecTor currently publishes 16 types of files. Files for bridges and relays have the same structure, but there are some differences with respect to the published information [107]. In particular, to avoid easy identification of bridges, their sensitive data is sanitized prior to online publication. The sanitization process includes the following 5 steps: (i) replacing the bridge's fingerprint with its SHA1 hash that we call *sanitized fingerprint*, (ii) removing most cryptographic information, (iii) removing bridge contact information, (iv) removing PT ports, and (v) replacing the bridge's IP address with a format-preserving sanitized version that we call *AIP*. This AIP is of the form 10.x.x.x, where x.x.x are the 3 most significant bytes of the hash SHA256(IP || fingerprint || secret); where "secret" is a 31-byte random string that changes once per month that is used to compute the AIP of all bridges during that month.

Next, we describe the 4 files we use in this paper: bridge server descriptors, network statuses, and extra-info descriptors for studying public bridges; and network status consensuses for identifying relays and filter them from our results.

Bridge Server Descriptors. These descriptors are produced by bridges and sent to the Bridge Authority. CollecTor publishes a sanitized version containing information such as the bridge's nickname ("Unnamed" by default), sanitized fingerprint, OR port, and AIP. They may also contain contact information of the bridge operator. Unsanitized bridge server descriptors can also be obtained from bridges' themselves (if their IP is known), by connecting to their OR port. Descriptors obtained directly from the bridge contain its fingerprint (rather than its sanitized fingerprint), the real IP address (rather than the AIP), and the bridge contact information (if provided).

Bridge Network Statuses. These files are produced by the Bridge Authority and capture which public bridges are available and their current status, so they can be distributed to users by BridgeDB. While some of their data is also available in bridge server descriptors (e.g., nickname, AIP, OR port), bridge network statuses do not contain the sanitized fingerprint, but instead include the bridge uptime and the flags it has been assigned by Tor's authorities (e.g., Running, High-Bandwidth). We use bridge network statuses to measure the bridge population and its stability. In particular we use the Running flag to determine if a bridge is *active*, and thus distributed to users. This flag is assigned to a bridge if and only if the Bridge Authority was able to reach the OR port using the vanilla Tor protocol in the last 45 minutes [107].

Bridge Extra-Info Descriptors. These files are sent to the Bridge Authority by the bridges approximately once a day (every 18h by default according to [107]). These files contain the nickname and sanitized fingerprint, the PTs supported by the bridge and usage statistics (number of IPv4 and IPv6 connections, number of unique IP addresses that have connected from a country, and number of unique IP addresses that have connected using each PT). Statistics are sanitized by rounding them to the nearest multiple of 8. Still, we can use them to rank bridges according to different criteria that may be relevant for an adversary.

Network Status Consensuses. These files contain information on which relays are available in the Tor Network, their status, and their endpoint information (IP address and OR port), so that they can be chosen by clients. We use these consensus files to differentiate relays from bridges.

4.3.2 Scan Search Engines

Scan search engines index data from Internet-wide scans on a number of target ports. Each port is scanned using a popular protocol on the port, e.g., TLS on 443/tcp or SSH on 22/tcp. We use scan search engines to identify IP addresses on the Internet that serve certificates matching a specific Tor pattern. We use two different scan search engines, described below.

Shodan. Shodan [72] scans over 200 ports using different protocols (e.g., TLS, SSH, HTTP, SMTP). When a supported protocol is identified on an IP, it indexes the service's text description (and the server certificate for TLS-based services). Among the scanned ports, 19 are scanned using a TLS handshake, which we can use for identifying IP addresses running Tor bridges (see Section 4.4.2). Once we identify a bridge, we query Shodan about data related to other services running in different ports of the same machine (e.g., SSH, HTTPS), in order to discover additional bridge IP addresses.

Censys. Censys [30] scans a smaller number of ports than Shodan using TLS (only 6) but more regularly, typically on a weekly basis. Similar to Shodan, for these 6 ports, Censys collects TLS handshake data, including the server's certificate, and it also publishes the raw scan data in addition to queries on the indexed information. We download the raw data from Censys and process it locally to identify IP addresses that run ORs and have their OR port in one of the 6 scanned ports.

4.4 Security Analysis Description

In this section we describe what properties we measure and the methodology used to perform those measurements. First, we introduce the measurements on public bridges performed using CollecTor data (Section 4.4.1). Then, we detail the approach we use to identify private bridges and proxies, (Section 4.4.2).

4.4.1 Public Bridges Analysis

We use the data provided by CollecTor to measure characteristics of the Tor public bridge population. Beyond understanding the demographics of the public bridges, our goal is to perform an in-depth analysis into how the fine-grained (i.e., per-bridge) publicly available data in CollecTor may impact the security of public bridges. One of the goals is to identify data that may need to be removed or to be sanitized prior to its publication. We evaluate the following 5 security-relevant properties:

(1) Bridge Population. We measure the number of bridges in the Tor Network in order to understand how large is the attack surface that an adversary needs to target for enumerating all bridges.

(2) Bridge Stability. We measure how stable bridges are in terms of lifetime and IP address changes in order to understand the vulnerability of bridges to aggressive blocking policies.

(3) Pluggable Transport Deployment. We measure the deployment of PTs over time in order to understand how long it takes to deploy a new PT and whether bridges offer multiple PTs with conflicting security properties.

(4) OR Port Distribution. We measure the frequency with which bridges use specific OR ports to evaluate how valuable this information is for an adversary that leverages the issues explained in 4.2.2 for discovering private bridges and proxies and for deanonymizing the IP address of public bridges.

(5) Bridge Importance. We rank bridges in terms of number of clients supported for different countries and PTs, showing that not all bridges are equally important. While some are rarely used, others represent vital elements in terms of the number of clients that connect to them, or the PTs they offer. Beyond improving our understanding of public bridges usage, we want to raise attention to how useful such rankings could be for an adversary. For instance, to evaluate how effective her bridge enumeration is for a given goal (e.g., "are all the bridges offering a particular transport identified?" or "are the top bridges for a country identified?"), or to target selected bridges, e.g., unblocked ones that are hard to identify because they run stronger PTs (e.g., obfs4, or ScrambleSuit).

To rank bridges, we extract usage statistics from CollecTor's extra-info descriptors. These are published periodically as introduced in Section 4.3, which means that rankings could reflect accurate real-time information and, even though they are sanitized by rounding to multiples of 8, they still allow to order bridges in terms of number of clients they serve.

4.4.2 Private Bridges and Proxies Analysis

The goal of our analysis is to gain a better understanding of the characteristics of the private bridge infrastructure in the Tor Network, e.g., population size, configuration, and hosting. In particular for *private proxies*, which are unknown to the Tor Project[3], we measure the following properties: the type of OR backend (i.e., relay, or public / private bridge) they forward traffic to, their configuration with respect to the backend (e.g., line of proxies to one backend, one proxy per

[3]This fact was confirmed in a private conversation with Tor developers

backend), the AS in which they are located and whether it is the same as the backend.

Since private bridges and proxies do not appear in CollecTor, studying them requires to first discover them on the Internet. We first describe our approach to identify hosts running private bridges and proxies, and then the clustering method we use to better understand their infrastructures.

Discovering Private Bridges and Proxies. We use a 5-step process to discover private bridges and proxies that leverages the open issues described in Section 4.2.2.

Step 1 – Finding candidate IP addresses. The first step consists on performing Internet-wide scans on a selected set of ports, starting a TLS handshake on each IP:port pair, and collecting the TLS certificate when the handshake succeeds. If the certificate collected from an IP address matches the pattern associated to Tor certificates described in Section 4.2.2, then it can be concluded that the IP address serving the certificate corresponds to a Tor OR (or a proxy to a Tor OR). To maximize the number of bridges identified with a limited scan budget, we leverage the OR port distribution that can be computed from CollecTor's data and focus on the top OR ports.

Since Internet-wide scans can be expensive to perform, and to avoid disrupting the Tor Network, we choose to substitute active scanning by queries to the Censys [30] and Shodan [72] scan search engines. Note that an adversary could similarly leverage such engines to minimize her scanning investment.

Step 2 – Filtering relays. The previous step produces a set of IP addresses running ORs (or proxies) at the time of the scan. Some of these IP addresses could correspond to Tor relays, which use the same kind of certificates as bridges. We use the Network Status Consensuses from CollecTor to classify IPs as relays. Any IP address that does not correspond to a Tor relay is a *discovered* IP address, i.e., running a Tor bridge (or proxy) at the time of the scan.

Step 3 – Verifying IP addresses. Next, our approach connects to the discovered IP address on the scanned OR port using the vanilla Tor protocol to try to download a bridge descriptor. If a descriptor is successfully downloaded, we say that the IP address is *verified*, i.e., still running a bridge (or a proxy). Furthermore, while Tor certificates are so distinct that we have not observed false positives from the regular expression used in Step 1, this step guarantees that there are no false positives since a verified IP address speaks the "vanilla Tor" protocol.

Step 4 – Identifying private proxies. To identify private proxies our approach compares the verified IP address from where a descriptor was collected in Step 3 with the IP address that appears in the content of the descriptor. A discrepancy between both IP addresses indicates that the verified IP address corresponds to a proxy that forwards traffic to a backend OR, to whom the descriptor belongs, running on the IP address leaked inside the descriptor. If no discrepancy is found,

the verified IP address corresponds to a bridge.

Step 5 – Classifying fingerprints. A downloaded descriptor contains the bridge unsanitized fingerprint, which can be hashed to obtain the sanitized fingerprint. We then search the sanitized fingerprint in CollecTor. If found, the descriptor belongs to a public bridge, otherwise it belongs to a private bridge. For public bridges, the mapping of an IP address to a specific bridge (i.e., sanitized fingerprint) in CollecTor, means access to all its historical data.

Discovery through non-Tor Services . Once a bridge is identified, it is possible to enumerate other services offered on the host by performing a vertical scan on its IP address seeking for open ports. Those additional services may provide unique identifiers (UIDs) such as SSH keys or TLS certificates that may enable discovering other bridges from the same owners, or tracking the bridge across IP changes. The vertical scan can be replaced by querying for the IP address in Shodan, since it already scans an IP on over 200 ports with popular protocols. Once UIDs are available, periodic queries to Shodan using those UIDs can be used to find new IP addresses where the UIDs have been observed. Once a candidate IP appears, Steps 2–5 above can be applied. For public bridges the OR port from where to try to download the descriptor can be obtained from CollecTor. For private bridges we first test the OR port of the bridge from where the UID was original found, in case it has not changed. Otherwise, Shodan is queried for open ports on the candidate IP address with a TLS certificate matching the Tor pattern. If that also fails, a vertical scan can be performed on the candidate IP (using TLS or the Tor protocol).

Clustering. To better analyze our results, we cluster public bridges, private bridges, and proxies into groups belonging to the same organization. Such clustering enables us to study the characteristics of bridge/proxy infrastructures in use. More precisely, we cluster tuples of (verifiedIP, port, descriptor) where verifiedIP and port correspond to the Internet endpoint from where the descriptor was downloaded. Our clustering uses 5 Boolean similarity features between tuples:
(1) *Same fingerprint.* Tuples with descriptors containing the same fingerprint come from the same bridge, regardless if collected from different verified IP addresses, and thus are similar.
(2) *Similar nicknames.* Nicknames are chosen by the bridge owner. Hence, we consider similar tuples with descriptors containing resembling non-generic nicknames. That is, two tuples are similar if the nicknames are identical and not generic (“Unnamed”, “default”, “anonymous”, “ididnteditttheconfig”, “ididedittheconfig”), or if they have the same prefix of length 5 or more characters that is not generic (“torat”, “relay”, “ec2bridger”[4]), e.g., mybridge3, mybridge4.

[4]This generic prefix is due to the now deprecated Tor Cloud image [106].

(3) *Same contact information.* The contact information is a free-text string selected by the bridge owner that often contains an email address, but may have other content. We consider similar tuples with descriptors with identical, non-empty, contact information.
(4) *Similar verified IP address.* This feature captures that similarly configured bridges on nearby IP addresses likely belong to the same owner. Tuples whose verifiedIP is in the same /24 subnet and for which the descriptors contain identical values for 5 fields (orport, socksport, dirport, Tor version, OS) are similar.
(5) *Similar IP address in descriptor.* Tuples whose IP in the descriptor is the same or that the IP is in the same /24 subnet and for which the descriptors contain identical values for 5 fields (orport, socksport, dirport, Tor version, OS) are similar.

Two tuples with at least one of the above features returning similar are placed in the same cluster. For each cluster, we obtain statistics such as the number of fingerprints, IP addresses, private bridges, public bridges, and proxies. We also compute statistics on the hosting ASes used in the cluster and study cluster ownership based on the contact information optionally available in the descriptors of the cluster's bridges.

4.5 Public Bridges Analysis

In this section we analyze the data published by CollecTor about public bridges regarding features that may impact the Tor bridge infrastructure security.

4.5.1 Bridge Population

We use CollecTor to compute the number of public bridges in the Tor Network. We uniquely identify public bridges and relays by their sanitized fingerprint, assuming that ORs change fingerprint infrequently (an assumption we validate in Section 4.6.1).

We split the sanitized fingerprints in CollecTor into *active* if they appear at least once with the Running flag (explained in Section 4.3.1) in a bridge network status in a month, and *inactive* otherwise. Figure 4.2 shows the evolution over time of the number of active (green bar) and inactive (red bar) sanitized fingerprints in the Tor Network. The bridge population significantly varies over time: it steadily grows from 2.8K active public bridges in July 2012 up to a maximum of 12.7K in July 2014, and starts declining in January 2015 falling to 5.3K by April 2016. We have had discussions with members of the Tor Project about this recent decline in bridge population, but the reason remains unclear.

The yellow middle bar represents a cluster of 3 bridges run by the same organization, that we call by their nickname, *Ki*, which change fingerprint up to once an hour (but keep their IP addresses stable, see Section 4.6). The *Ki* clus-

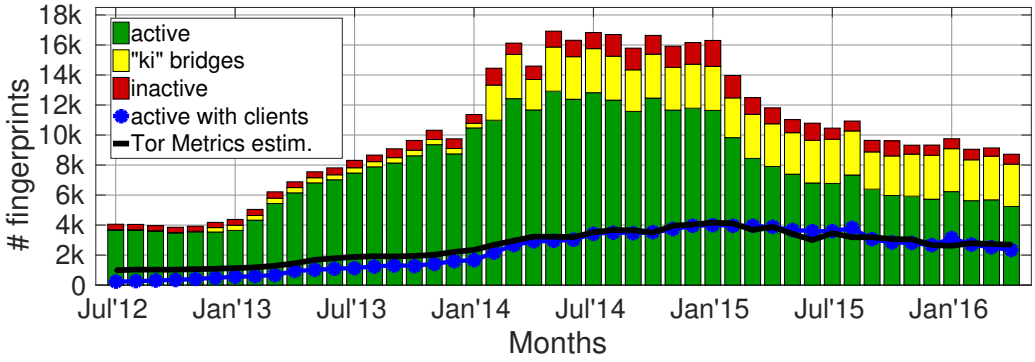

Figure 4.2: Number of active (bottom bar in green), inactive (top bar in red), and *Ki* (middle bar in yellow) sanitized fingerprints over time. The two lines correspond to the monthly average number of bridges reported by TorMetrics (black line) and the number of bridges with clients (blue line).

ter produced a few dozen fingerprints in July 2012, jumped to a few hundreds in December 2012 and to a few thousands in February 2014. In March 2016, those 3 bridges are responsible for 32% of all fingerprints, corresponding to 7% of the active fingerprints and 68% of the inactive fingerprints, as most of their fingerprints do not live long enough to obtain the Running flag. After discounting those extraneous fingerprints, the number of active fingerprints in April 2016 is slightly over 5K.

The figure also shows two lines representing the monthly average number of bridges reported by TorMetrics [110] (black); and the number of active bridges with at least one client (blue). These two values are very close to each other, though not the same. The blue line represents less than 50% of the active bridges in a month, indicating that more than half of the active bridges do not serve users. We examine this discrepancy in the next subsection.

We use both the number of active bridges (bottom green bar) and the number of active bridges with at least one client (blue line) as different baselines for other measurements.

4.5.2 Bridge Stability

In this section we use CollecTor to study how stable public bridges are by first measuring their lifetime (i.e., for how long they are active) and then how often they change their IP address changes over their lifetime.

Bridge Lifetime. We define the lifetime of a bridge as the time window when the bridge was active, i.e., the time difference, in days, between the last and first time a descriptor for an active bridge becomes available in CollecTor. Figure 4.3 captures the CDF of the bridges' lifetime in our study period for all active bridges

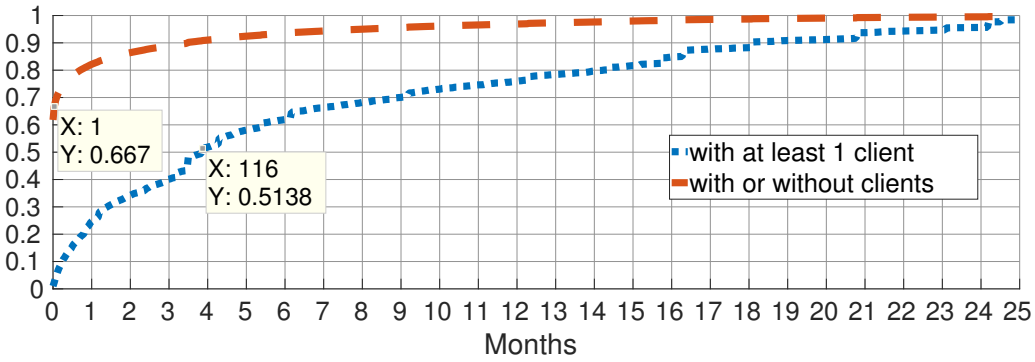

Figure 4.3: CDF of bridge lifetime over time (Jul'12 - Apr'16) for all active bridges (red dashed), and bridges that have had at least one client (blue dotted).

(red dashed line) and for bridges with clients (blue dotted line). We see that 67% of the active bridges live for less than one day, and thus are unlikely to be used by clients. This explains the difference between the bridges with clients and the active bridges in Figure 4.2. However, bridges that are used by clients are quite stable. Their median lifetime is 116 days (roughly 4 months) and 25% of them live over one year.

Bridge IP Changes. Next, we evaluate how often public bridges change IP address. As explained in Section 4.3.1, bridges' IP addresses are replaced in CollecTor with its AIP for privacy reasons. Yet, the AIP construction algorithm, which every month assigns new AIPs to bridges associated to the monthly secret, allows to compute the number of IP changes. It suffices with counting the number of AIPs assigned to a bridge's fingerprint and subtract the number of month changes in its lifetime.

Figure 4.4 shows the CDF for the number of IP addresses for all active bridges (black line) and for bridges with at least one client (blue line). The figure shows that 67% of the active bridges have a single stable IP. This number grows for bridges with clients where 84% of bridges never change IP address, and 90% had at most one IP address change.

These results show that 55% of the bridges IPs are short-lived and thus these bridges do not carry users. On the other hand, bridges that do carry users are quite stable. They live for roughly 4 months and 84% of them never change IP address. These results have important implications for a censor: they show that current policies that remove blocks for bridge IP addresses every 25 hours [37] are extremely polite and adversaries could be performing a more aggressive blocking (up to months), without the risk of creating too many false positives.

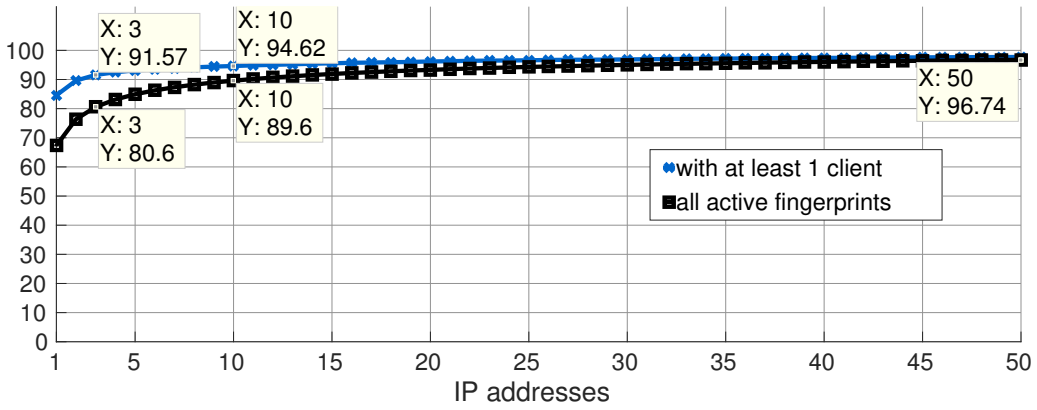

Figure 4.4: CDF of number of bridge IP addresses for all active bridges (bottom) and for bridges with at least one client (top).

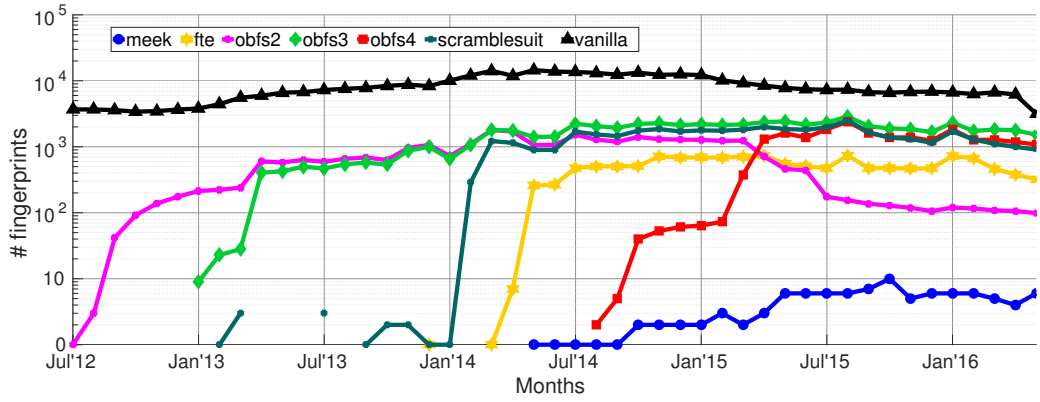

Figure 4.5: Number of active fingerprints offering each transport over time (note the y-axis logarithmic scale).

4.5.3 Pluggable Transports Deployment

We now examine PT deployment across time and bridges. Figure 4.5 depicts the number of active public bridges, in logarithmic scale, offering each transport over time. It indicates that the most popular transport is vanilla Tor. Although its popularity has started to slowly decrease, it is still offered by 77% of all bridges in April 2016. The timeline shows how the deployment of obfs4 coincides with the decline of the deprecated obfs2 [9]. It also shows that after a PT is introduced, it takes between 4 months and one year to reach a stable number of 1K–2K bridges offering it. Surprisingly, deployment of different PTs does not improve beyond that stability point, an issue that we examine next.

A bridge can offer multiple transports. Table 4.1 shows the most popular transport combinations in April 2016. Rows in gray highlight that a single transport is offered. Surprisingly, 77% of the bridges only offer vanilla Tor, a transport

vanilla	fte	obfs2	obfs3	obfs4	meek	ssuit	Bridges
✓	-	-	-	-	-	-	6,213 (77.1%)
-	-	-	✓	✓	-	✓	524 (6.5%)
-	-	-	✓	-	-	-	510 (6.3%)
-	✓	-	✓	✓	-	✓	353 (4.4%)
-	-	-	✓	✓	-	-	242 (3.0%)
-	-	-	✓	-	-	✓	129 (1.6%)
-	-	-	-	✓	-	-	117 (1.4%)
-	-	✓	✓	-	-	-	72 (0.9%)
-	-	✓	✓	✓	-	-	27 (0.3%)
-	-	-	-	-	-	✓	22 (0.3%)
-	✓	-	✓	✓	-	-	20 (0.2%)
-	✓	-	-	-	-	✓	6 (<0.1%)
-	-	✓	-	-	-	-	5 (<0.1%)
-	✓	-	-	-	-	-	4 (<0.1%)
-	-	-	-	-	✓	-	3 (<0.1%)
-	✓	-	-	✓	-	-	2 (<0.1%)

Table 4.1: Most frequent transport combinations in April 2016. Combinations offered by a single bridge or by inactive bridges are not included.

that is trivial to identify through traffic analysis. The 1K–2K bridges offering obfs3, obfs4, and ScrambleSuit (ssuit) responds to the deployment of multiple transports on the same bridges.

The combination of PTs with different security properties raises several security concerns, since the security of the bridge is only as strong as its weakest link. First, an adversary detecting the weakest transport and blocking the IP disables also stronger transports for free, e.g., for the nearly 100 bridges that offer obfs3 or obfs4 in combination with obfs2, which is deprecated and trivial to identify through traffic analysis. Second, it allows an adversary to confirm a bridge, even in presence of transports that implement reply protection. For example, for the most popular combination obfs3+obfs4+ScrambleSuit, offered by 524 bridges, an adversary can confirm a bridge, e.g., identified through traffic analysis [119], through a vertical scan using obfs3 on the candidate IP address.

4.5.4 OR Port Distribution

In this section we use CollecTor to find the most common OR ports employed by public bridges. If an adversary knows that the majority of bridges are running on a few OR ports she can use them as targets for deanonymization via Internet-wide scanning, as described in Section 4.4.2.

First, we study the stability of a bridge's OR port, i.e., how often bridges change their OR port. We find that 99% of active fingerprints never change their OR port during their lifetime.

Next, we study the OR port distribution. During our observation period, bridges used 7,985 different OR ports. However, we observe that four ports (443/tcp, 8443/tcp, 444/tcp, and 9001/tcp) are chosen much more often than the rest, while all other OR ports are used only by a small subset of bridges.

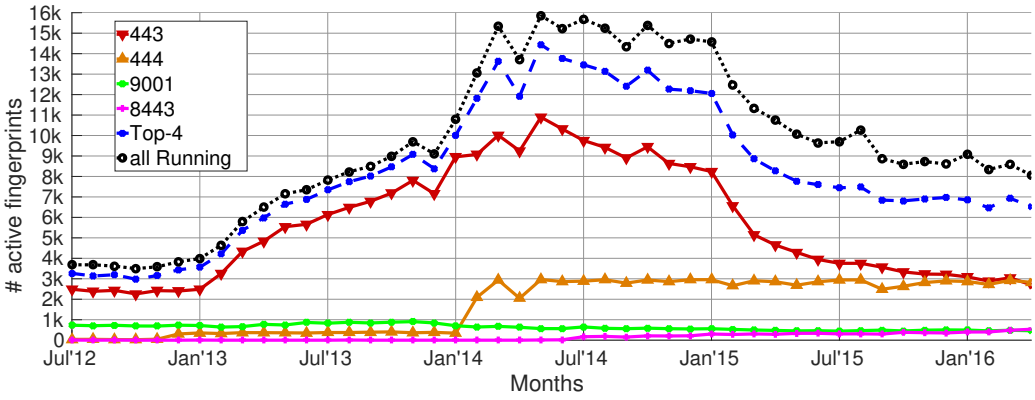

Figure 4.6: Top 4 OR ports used by active public bridges over time. The top dotted line represents all active public bridges; the dashed line below corresponds the Top-4 OR ports together; and each solid line represents one OR port.

Figure 4.6 reports the OR port usage over time. The top line corresponds to the total number of active fingerprints. The dashed (blue) line directly below corresponds to the top 4 OR ports aggregated. Each solid line below corresponds to one of the top 4 OR ports. On average, the top 4 OR ports are used by 82% of the fingerprints observed each month, although this fraction has decreased from 95% in March 2013 to 82% in April 2016. The most common port is 443. We conjecture that its popularity is due to two main reasons: it is the default HTTPS (HTTP over TLS) port, which makes it less likely for TLS-looking vanilla Tor traffic to stand out, and it is a port typically open in firewalls so it can be reached by most users. We assume that for similar reasons 8443, the alternative HTTPS port, is the second most popular port. The third most popular port, 9001, is the standard Tor port.

Port 444 is a special case since in principle is associated to the Simple Network Paging Protocol (SNPP), a not so popular protocol. However, according to CollecTor data, roughly 3K active fingerprints are using it on April 2016. The reason for this is that this OR port is used by the *Ki* bridges that change fingerprint often, as introduced in Section 4.5.1. Those *Ki* bridges artificially inflate the usage of this OR port, a behavior that does not manifest on other OR ports.

In summary, the OR port distribution shows that an adversary could deanonymize 71% of all active public bridges by scanning 3 OR ports (443, 8443, 9001) and 82% if we consider the Top 4 with the anomalous 444. While 443 usage has declined over time, the top 4 OR ports usage closely follows the total active bridges, thus OR port diversity does not seem to be improving over time.

After we reported our findings to the Tor Project, they opened a ticket to sanitize the OR port in CollecTor [69] so that our experiments cannot be replicated.

4.5.5 Bridge Importance

As introduced in Section 4.3.1, not all bridges are equally important and CollecTor can be used to rank bridges according to different metrics. We now evaluate two example scenarios: ranking bridges by country usage and by PT usage. These two rankings are very relevant to, for instance, a censor that wants to know how many (or which) bridges have to be blocked to minimize the likelihood of users connecting outside the country; or to block a strong PT that it cannot be identified through traffic analysis or active probing. For our evaluation we select 5 countries: three where Internet censorship is known to occur [53]: China (cn), Iran (ir) and Syria (sy); and two where monitoring is a real threat: United Kingdom (uk) and United States (us) [100, 129].

Ranking per country. Table 4.2 summarizes the ranking per country for April 2016. For each country it shows the country code, the number of bridges that received at least one connection from users located inside the country, the average number of clients per day served by those bridges, the percentage of these clients served by the Top 20 bridges, with the fraction of served by default bridges[5] in parentheses, and the total traffic carried by default bridges. For the last two columns the number of default bridges is shown in square brackets. The *All* row at the bottom corresponds to statistics for all bridges in CollecTor regardless of user location.

The Top 20 bridge statistics show that a few bridges handle the majority of clients. The Top 20 support more than 91% of traffic for the full Tor Network, and more than 75% for all countries except China and the US, where clients are better spread among available bridges. The Top 20 is dominated by default bridges. In fact, the Top 14 bridges in the full Tor Network, and also in all countries except the US, are default bridges. Thus, we can conclude that default bridges carry the vast majority of clients in the Tor Network.

This dominant choice of default bridges, which are trivial to deanonymize through the configuration files shipped int the Tor client software, means that in the majority of cases clients route their traffic through known bridges, thus defeating the very purpose of bridges.

Ranking per PT. Table 4.3 summarizes the ranking per PT for April 2016. Its structure is the same as that for Table 4.2. The results indicate that while vanilla Tor is the most widely deployed transport, most clients opt for more recent PTs such as obfs4, obfs3, and meek. ScrambleSuit (ssuit) is not very popular and, as expected [9], obfs2 has almost completely stopped being used.

We observe that the Top 20 bridges handle almost the entire user base for obfs4, obfs3, and meek. The vast majority (90%–100%) of clients using these 3 transports are carried by default bridges. The Top 20 bridges do not dominate

[5]We use the default bridges available in the Tor Browser 5.5.5 distribution.

CC	Used Brid.	Clients	Top 20 (Default)	Total Default
cn	712	4,265	45.6% (44.0%) [14]	44.3% [18]
ir	941	26,479	86.6% (86.1%) [16]	86.1% [18]
sy	74	449	76.9% (68.0%) [14]	69.2% [17]
uk	943	16,723	84.1% (84.0%) [17]	84.0% [19]
us	1,496	17,911	58.7% (56.7%) [6]	56.9% [11]
All	2,213	301,009	91.71% (91.4%) [17]	91.4% [23]

Table 4.2: Bridge importance per country (Apr'16)

PT	Used Brid.	Clients	Top 20 (Default)	Total Default
vanilla	1,967	14,939	5.6% (0.0%) [0]	1.2% [21]
obfs2	13	158	100.0% (25.8%) [1]	25.8% [1]
obfs3	898	63,088	92.0% (90.8%) [4]	90.8% [4]
obfs4	792	204,095	95.4% (94.7%) [11]	94.7% [11]
ssuit	467	4,483	52.4% (46.3%) [1]	46.3% [1]
meek	4	22,685	100.0% (∼100%) [3]	∼100% [3]

Table 4.3: Bridge importance per PT (Apr'16).

vanilla Tor and ScrambleSuit, likely related to no default bridge being marked as offering "vanilla Tor" in the Tor client configuration files and only one offering ScrambleSuit. Still, the lone default bridge offering ScrambleSuit carries over 46% of the clients using that transport.

OR Port Distribution per Country. In Section 4.5.4 we have studied the global ranking of OR ports and have shown that the top 3 OR ports are used by 71% of all public bridges. Here, we analyze whether the top bridges by number of clients also follow that distribution.

Table 4.4 contains the Top 20 OR ports by percentage of clients served in the full Tor Network. For each OR port it reports the rank, the percentage of clients supported by bridges using that OR port, the number of bridges using that OR port (and how many are default bridges in square brackets), and the ranking of the OR port when only considering connections from a particular country.

The table shows that the choice of OR port among popular bridges does not resemble the distribution of OR ports in the overall bridge population. The Top 6 OR ports correspond exclusively to default bridges. The first OR ports in the Top 10 global distribution appear at rank 8 (9002) and rank 10 (9001). In both cases, the majority of the clients served on those ports correspond to default bridges, e.g., for port 9001, the percentage of clients served by the 303 non-default bridges is negligible. From that point, we see that the most popular bridges run in random

RK	Port	Clients (%)	BRs [Default]	Ranking per country cn	ir	sy	uk	us
1	6666	23.805%	1 [1]	2	5	6	1	1
2	42506	14.096%	1 [1]	6	3	4	3	-
3	60906	13.877%	1 [1]	7	4	3	2	-
4	63848	13.730%	2 [2]	5	6	5	4	4
5	44445	9.485%	1 [1]	8	2	2	5	2
6	8008	7.173%	1 [1]	4	54	-	6	-
7	29001	5.027%	2 [1]	10	1	1	7	3
8	9002	2.827%	2 [1]	1	7	8	8	-
9	1512	1.206%	1 [1]	3	8	14	9	125
10	9001	0.263%	309 [6]	19	9	7	10	5
11	29309	0.045%	1 [0]	36	10	-	42	10
12	27134	0.041%	1 [0]	15	13	18	12	16
13	20506	0.040%	1 [0]	59	19	19	11	7
14	12497	0.040%	1 [0]	57	14	-	42	9
15	59760	0.039%	1 [0]	18	19	-	33	11
16	60841	0.039%	1 [0]	49	15	-	50	16
17	53885	0.038%	1 [0]	15	36	-	50	14
18	14769	0.035%	1 [0]	38	61	-	11	6
19	34678	0.033%	1 [0]	37	12	-	66	8
20	19924	0.032%	1 [0]	12	19	-	19	14

Table 4.4: OR port ranking for most used bridges (Apr'16).

high ports. These results suggest that owners of non-default popular bridges are careful to set the OR port selection to random, but those of less popular bridges are not as careful in this respect.

The ranking per country shows that, even though in general default bridges dominate in all countries, the popular non-default bridges can vary significantly across countries. For example, in Syria the globally popular bridges have little overlap with the popular ones in that country. This suggests that a state-level adversary can use CollecTor to compute a ranking of, currently unblocked, OR ports that she should target next through Internet-wide scanning to maximize the blocked population in her country.

Another observation (not shown in Table 4.4) is that all Top 20 non-default bridges incur in the problem flagged in Section 4.5.3 of offering multiple PTs with different security properties.

Port	SC	Source	Scan Dates	Verif. Date	Disc.	Verified	Public	Private	Proxy
443	9	Censys	04/04–04/28	04/08	2,448	1,315 (1,122)	897 (860)	263 (262)	164
993	2	Censys	04/20–04/27	04/21	19	16 (13)	11 (11)	3 (2)	2
995	3	Censys	04/15–04/29	04/23	14	14 (13)	10 (10)	3 (3)	1
444	1	Shodan	04/19–04/19	04/19	14	12 (101)	8 (97)	1 (4)	4
8443	1	Shodan	04/21–04/21	04/22	191	156 (149)	148 (148)	1 (1)	7
9001	1	Shodan	04/17–04/17	04/18	2,001	1047 (587)	165 (166)	415 (421)	468
9002	1	Shodan	04/23–04/23	04/23	23	19 (5)	1 (1)	4 (4)	14
All	17	All	04/04–04/29	04/08	4,684	2,554 (1,986)	1,239 (1,292)	684 (694)	645

Table 4.5: Bridge discovery in April 2016

4.6 Private Bridges and Proxies Analysis

In this section we analyze private bridges and proxies in the Tor Network. Section 4.6.1 presents the results of applying the bridge discovery approach described in Section 4.4.2 on the Tor Network during April 2016, Then, we analyze the discovered bridge/proxy infrastructures (Section 4.6.2).

4.6.1 Discovering Private Bridges & Proxies

Since we do not run our own scans, we can only apply the bridge discovery to 7 out of the top 10 OR ports in April 2016, which are scanned by either Censys or Shodan using TLS. Table 4.5 summarizes our results. For each OR port we report: the number of scans available on that port in April 2016 (SC), the source of the scanning data (either Censys or Shodan), the first and last scan dates, the date when verification started, the number of discovered bridge IP addresses, i.e., those where scan data shows a certificate matching the Tor pattern and that are not relays, the number of verified IP addresses (and fingerprints in parentheses) from which we were able to download a bridge descriptor, and the split of verified IPs (and fingerprints) into public bridges, private bridges, and private proxies. Note that we distinguish public and private bridges through fingerprints, but proxies by IP address, as proxies are not Tor ORs and have no fingerprint. The last row shows the aggregate results for all ports.

Overall, we discover 694 private bridges and 645 private proxies, which do not appear in CollecTor. Additionally, we deanonymize the IP address of 1,292 public bridges. According to CollecTor data, these correspond to 35% of public bridges with clients (23% of all active public bridges) in April 2016, excluding *Ki*-related bridges. On the 7 OR ports examined, the bridge population comprises 65% public and 35% private bridges, i.e., one in three bridges is private.

There exist several reasons why we do not deanonymize a larger fraction of public bridges in CollecTor. First, we can only deanonymize bridges on the 7 OR ports scanned by Censys or Shodan. Second, for most ports we only have one scan in the second half of the month. Hence, we cannot discover short-lived bridges active only in the first half. Furthermore, we have shown that 55% of the active bridges are short-lived; thus, while they may appear as candidate IPs, we may not be able to confirm them because they no longer live when we try to download a descriptor from them.

We note that the aggregated results differ slightly from the sum of all rows for two reasons. First, a few proxies have more than one OR port open and thus their IPs are counted in more than one row. In addition, a few IP addresses were observed hosting a proxy at some point in the month and hosting a bridge at other times. Also, note that the number of verified fingerprints for port 444 is significantly larger than the number of verified IPs. This is because 6 of these IPs (3 proxies and 3 public bridges) belong to the *Ki* cluster that changes fingerprint

periodically.

Discovery through non-Tor Services. Next, we evaluate if additional services running on bridge hosts can be leveraged to track bridges across IP address changes. For this, we assume an adversary discovered a bridge by any means, (e.g., the approach in Section 4.6.2, querying BridgeDB, or adding a middle OR in the Tor Network [66]), but the open OR port issue we leverage has been solved by the Tor Project.

First, we measure the percentage of bridges in Table 4.5 with additional ports open (beyond the OR port). For this, we query each verified IP address at Shodan. Overall, 621 (24%) of the 2,554 verified IPs offer at least one additional service (beyond the OR port) and 10% more than one. In total, we observe 101 additional ports. These numbers indicate that it is not uncommon to run other services on a Tor bridge. The most common additional services are SSH on ports 22 and 2222, Web services on ports 80 and 443, and RPC port mapper on 111. As unique identifiers (UIDs), we use SSH keys on ports 22 and 2222 and certificate serial numbers on 443.

Scanning those UIDs in Shodan on May 18 provides us with 2,248 candidate IPs, of which only 248 return a descriptor. After filtering out relays and already known IPs, we found 9 new bridge IP addresses that were not observed in April 2016. For example, one of them is located in Amazon EC2 and it corresponds to a bridge that was running on another IP in April 2016, but likely changed IP because the EC2 instance was re-started, as EC2 assigns VM IP addresses from a shared pool.

Fingerprint Stability Validation. The discovery of bridges, which provides us with access to their unsanitized descriptors, allows us to validate the assumption that OR fingerprints rarely change, and thus are indeed good bridge identifiers. We periodically (roughly once a day until June 3rd, 2016) try to download a descriptor from verified IP addresses. Then, we measure the frequency of fingerprint changes in the descriptors for bridge (i.e., non-proxy) IPs found in April 2016.

Overall, 94.1% of the bridge IP addresses did not change fingerprint, 5.5% changed fingerprint once, and 0.4% changed fingerprint multiple times. The bridges with multiple fingerprint changes include the 3 *Ki* bridges, which present a different fingerprint every time we connect to them (on a closer look we find that they change fingerprint roughly every hour). Furthermore, we observe that over 70% of the IP addresses with fingerprint changes belong to 2 clusters of private bridges each using multiple nearby IP addresses. These IPs change fingerprint on the same dates, so it is possible that bridges in each cluster were reassigned IP addresses on those dates.

These numbers confirm that the vast majority of bridges do not change fingerprint over time. Thus, bridge fingerprints (and sanitized fingerprints in CollecTor)

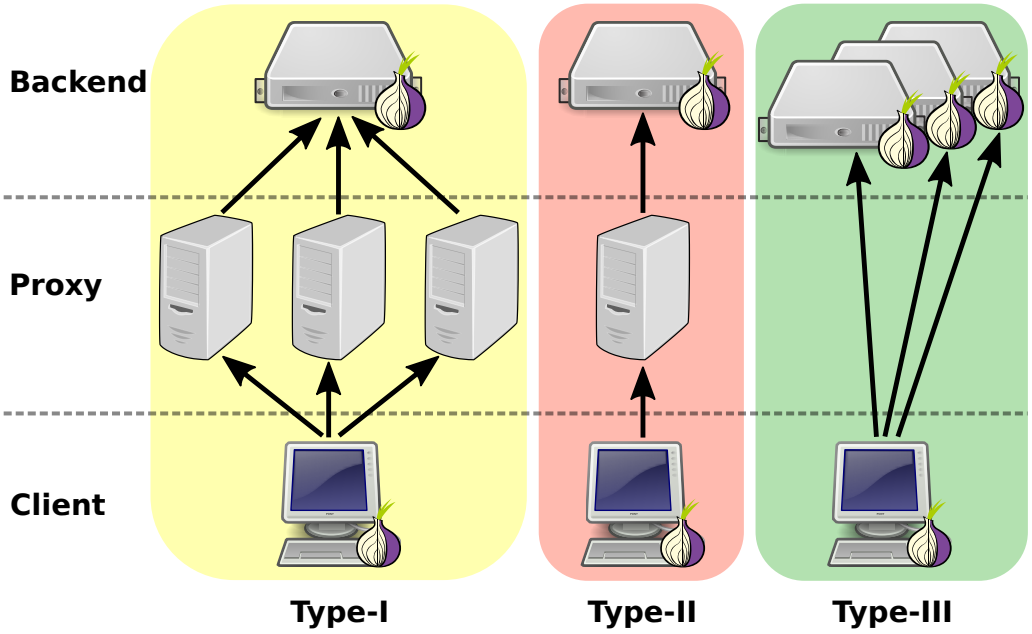

Figure 4.7: The three most common cluster types.

can be used as identifiers.

Hosting. The discovered proxies are hosted in 166 ASes, the private bridges in 139, and the public bridges in 385. The top 10 ASes for both bridges and proxies are dominated by the ASes used by the top clusters we find in the next Section. For bridges, the top 10 ASes comprise: 5 popular cloud hosting providers, 2 broadband residential ISPs, and 3 large ISPs that provide multiple services. While overall there seems to be enough IP diversity among the bridge population, in the next section we show that individual clusters exhibit less diversity.

Contact Information. Bridge descriptors may contain contact information that could reveal ownership. We find 267 email addresses in the descriptors collected from the 1,986 public and private bridges. Of those, 69 have a domain name from a public email service provider (e.g., Gmail, Yahoo), and thus cannot be easily used to establish ownership. The other 198 email addresses contain 191 domain names, of which 175 return a valid mail server IP address through a DNS MX query, the rest we discard as invalid. Those 175 domains appear in descriptors from 307 bridges: 187 public and 120 private. This indicates that, contrary to what could be expected, private bridges often provide contact information that may reveal the organizations behind the bridge. This enables an adversary to identify Tor clients connecting to the bridge as members of that organization.

4.6.2 Bridge/Proxy Infrastructures

We cluster bridges and proxies as described in Section 4.4.2 to better understand how they are used. Out of 41,359 tuples (verifiedIP, port, descriptor), collected by connecting periodically to the 2,554 verified IP addresses, we obtain 1,343 clusters, of which 75% are singletons, i.e., contain a single bridge and no proxies.

We identify 5 cluster types, as well as mixed clusters with subclusters of those types. The 3 most common cluster types are shown in Figure 4.7. Type I corresponds to a line of proxies (from 2 up to 178) that all forward to the same backend. Type II is a Type I cluster with a single proxy. For both types, the backend can be a private bridge, a public bridge, or a relay. Type III is a cluster of multiple bridges belonging to the same owner, without proxies. Type III clusters can have only public bridges or only private bridges.

Not shown in Figure 4.7 are the rarer Type IV and Type V. Type IV corresponds to the same bridge running on multiple IPs *at the same point in time*. This differs from a bridge changing IP over time where we would observe the same fingerprint at different IPs on different days. This cluster type could be a result of cloning an image with an installed bridge on multiple VMs. Type V corresponds to a proxy load-balancing across multiple backend IPs. In most cases, the backend IPs host the same bridge (fingerprint), which runs on a Windows host. This is similar to a fast-flux network where residential hosts run bridges, and the proxy provides IP stability to access them.

We develop a set of rules to automatically classify non-singleton clusters. The cluster classification outputs: 47 clusters as Type I (37 relay, 7 public, 3 private); 138 as Type II (94 relay, 29 public, 15 private); 88 as Type III (69 public, 19 private); 43 as Type IV; 10 as Type V; and 16 as mixed clusters containing at least two subclusters of other types.

Type I clusters most often have proxies forwarding to a relay (37 clusters) or a public bridge (7 clusters). A similar trend applies to Type II clusters. In all Type I clusters all proxies in the cluster are hosted in the same AS, and are often located in nearby and even consecutive IP addresses. In 77% of Type I and Type II clusters, the proxies and the backend are in the same AS. The cases where proxies and backends are in different ASes could indicate organizations free-riding on public ORs to perhaps avoid configuring their own bridge. We observe only 3 Type I clusters with private backend indicating that, since proxy IPs are already private, there is no benefit on a private backend.

In Type I and Type II clusters, proxies seem to be used by owners of public bridges and relays to keep a few IPs (only one for Type II) private for their own use. Those proxies do not provide much IP diversity; once a proxy is known, an adversary could scan the nearby IP addresses to find other proxies and the backend. Furthermore, when the backend is a relay, an adversary could scan IP ranges hosting relays to try to find proxies forwarding to it. It is also important to note that when the backend is a relay only "vanilla Tor" is supported, as relays

#	Type	IPs All	IPs Proxy	Bridges Pub.	Bridges Priv.	ASNs	Contact
1	I public	179	178	1	0	1,1,✗	✓
2	III private	164	0	0	164	0,1,✗	✓
3	I relay	72	71	0	0	1,1,✓	-
4	III private	63	0	0	63	0,1,✗	-
5	III public	53	0	53	0	0,16,✗	✓

Table 4.6: Statistics on top 5 clusters.

do not run PTs.

Type III clusters most often comprise public bridges (69 clusters) but can also have private bridges (19 clusters). In 93% of Type III clusters all bridges are in the same AS. This indicates that once an owner has established a relationship with a hosting provider, it is easier to install multiple bridges on that provider. Still, there are a few Type III clusters with good diversity. For example, the cluster at Rank 5 has 53 public bridges on 16 ASes. We observe that some Type III public clusters belong to organizations that also contribute relays to the Tor Network. Another public cluster belongs to a computer security company.

The mixed clusters show that our clustering can capture multiple infrastructures from the same owner. An example of a mixed cluster is the `Ki` cluster, which comprises a Type I subcluster with 2 proxies forwarding to a public bridge, another Type II subcluster with one proxy forwarding to one public bridge, and a singleton subcluster with a public bridge. Both bridges and proxies in the cluster keep their IP address stable, but bridges change fingerprint hourly.

We conclude that proxies are not generally used for load-balancing across multiple backend ORs as there are only 10 Type V clusters compared to 185 Type I and Type II clusters. Proxies do not seem to be used for OR port diversity either. We observe only 13 proxy IPs forwarding on multiple ports. Of those, 10 belong to 2 clusters where the same backend IP runs two different bridges on different ports and a proxy port forwards always to the same backend OR port.

Top Clusters. Table 4.6 provides details for the Top 5 clusters. For each cluster, it shows the rank by number of IP addresses; the type; the total number of IP addresses; the number of proxy IP addresses; the number of public and private bridges (by fingerprint); the hosting as a (x,y,flag) tuple with number of autonomous system numbers (ASNs) for proxy IPs (x), bridge IPs (y), and whether those ASNs are the same (✓) or not (✗); and if the bridges in the cluster contain contact information.

4.7 Costs

In this section we evaluate the costs, both in term of time and computational resources needed for carrying your deanonymization attacks. Bridge descriptors and "Network Status Consensus" consist only of a few hundreds of Megabytes and can be easily downloaded from CollecTor. The data is then processed an converted into a serialized dictionary for making faster the access to this information. Processing stats for one entire month, can be done in less than one hour. Censys scans are huge ".json" files, typically exceeding the size 100 Gigabytes. The download of a full scan takes roughly 24 hours. To process the data and identify certificates that have the features mentioned in Section 4.2.2, our script requires a few hours. Access to Shodan data is performed through dedicated APIs. Depending on the number of hosts that have a specific feature, obtaining information for all services running on a particular IP, can take from 1 up to 8 hours. Finally, validation requires to attempt to connect to each IP using "vanilla Tor"; only if this step succeeds and we are able to download the descriptor, we confirm that the IP belongs to a bridge. Attempting to connect sequentially to each candidate, using a single machine and with only one Tor client, on average can require up to 8 hours.

From the perspective of resource usage, the download and the analysis of data can be performed using a single machine, with no demanding requirements in terms of resources (CPU and RAM). For this reason the bottleneck for the whole process is represented by the download of the Censys scan. Because of this limitation attacks can be performed with a delay of a one day and half (i.e., the time required to download and process the most recent Censys scan in addition to the time to try to connect to each IP of a possible bridge).

4.8 Security Discussion of Findings

In this section we recapitulate the findings described throughout the paper and discuss their implication with respect to the security and privacy of Tor's bridge population.

4.8.1 Security Implications of Scan Search Engines

While the use of Internet-wide scanning for discovering bridges was known [31, 117], our work illustrates how the availability of scan search engines, which index data collected from Internet-wide scans, greatly lowers the attack cost in terms of scanning infrastructure. In fact, by leveraging search engines, an adversary that follows our approach can discover, with no investment in scanning infrastructure, 35% of public bridges with clients, 23% of active public bridges, and hundreds of private bridges and proxies. The discovered public bridges support 95% of all clients and include 90% of the bridges offering obfs4 and obfs3. In contrast,

other bridge enumeration attacks, e.g., using a middle node or performing clever queries to the Bridge Authority [66], achieve lower coverage, are much slower, and may require setting up an OR.

Furthermore, we have shown a novel technique to leverage scan search engines to discover Tor bridges by examining non-Tor services running on bridge hosts. Those services may provide unique identifiers (e.g., SSH keys) that may reveal other bridges from the same owners and enable tracking a bridge across IP changes.

One key take away is that scan search engines have greatly lowered the bar with respect to the resources needed to learn what services are offered on Internet-connected hosts. The security of systems that require hiding components, e.g., Tor bridges, should be designed with this threat in mind.

Even in case IPv6 was widely adopted, the situation would be slightly better despite the presence of scan search engines. The address space of 2^{128} possible IP addresses, would not allow scanners to collect data on a daily basis, but results obtained by querying one of those engines, would return the address of a bridge that was active several months ago. Adoption of IPv6 by itself would not be sufficient: stable bridges have a median lifetime of 4 months and they could still be identified even using information obtained from non-fresh scans.

Closing the OR Port. Having the OR port open in all bridges enables bridge enumeration through Internet-wide scanning. Closing the OR port has been queued for fixing since November 2012. While its priority was increased in September 2015 [7], the fix has not happened yet, as it requires changes to the Bridge Authority, BridgeDB, and multiple other tools that assume bridges have an OR port. Our quantification of the impact of this bug highlights the importance of quickly fixing vulnerabilities. We hope our work will provide a push to fix this bug and possibly to find an entity that sponsors the fix.

While the bug is fixed, possible stopgap fixes include allowing only known IPs (e.g., Bridge Authority) to connect to the OR port and improving OR port diversity. The latter is critical because closing the OR port by default does not solve the problem that it would remain open for the 77% bridges offering "vanilla Tor", which would still be vulnerable to scanning attacks. Non-default important bridges carrying a significant number of users seem to already be randomizing the OR port by setting it to `auto` in their configuration files, likely due to the expertise of their owners. This forces an attacker to scan many ports before finding a significant fraction of bridges. However, we have shown that 71% of public bridges use 3 OR ports. Thus, it is critical to educate less expert users on the importance of setting the OR port to `auto` in their bridge configuration. Bridge configuration tutorials that still use a fixed OR port, e.g. [79], should be fixed.

We must stress that closing the OR port does not eliminate the threat of discovering bridges through additional non-Tor services coexisting on a bridge's

host, nor other security risks uncovered by our analysis that we discuss in the following sections.

4.8.2 Security Implications of CollecTor Data

While collecting fine-grained data about the Tor Network is fundamental for the Tor Project maintainers, making it publicly available through services like CollecTor requires an analysis of the benefit versus privacy trade-off to understand the risks. One of our goals was analyzing if the fine-grained, per-bridge, data provided by CollecTor, as opposed to aggregate statistics provided by Tor Metrics, could be harmful to the security of public bridges. Here we discuss our findings.

OR Ports. The most concerning data in CollecTor we found is the availability of OR ports. Knowing the OR port of all public bridges helps to optimize the discovery of bridges by focusing Internet-wide scans on the most popular OR ports, which are used by the overwhelming majority of bridges. Furthermore, as discussed in Section 4.5.5, an adversary can search for the OR port of a bridge of interest (e.g., one supporting most users in a censored country or providing strong PTs) and then perform an Internet-wide scan on that OR port to deanonymize it. After we sent a draft of this work to the Tor Project, CollecTor has started sanitizing the OR port in a similar way as they anonymize bridge IPs [69]. Such sanitization prevents mapping an anonymized OR port in CollecTor to a real port, preventing the two issues above. Techniques that apply differential privacy would not have been enough in this situation, since from the anonymized data it would still be possible to infer correlation and usage of a small number of OR ports. Even if the attacker did not have access to the information about the most popular OR ports in CollecTor, she could still interact with BridgeDB and obtain bridge samples with the corresponding OR port on which they are listening. By knowing the only the shape of the distribution of the ports, the sampled dataset can be leveraged by the adversary for inferring which should be the most popular OR ports, to be selected during scans, that would maximize the likelihood of discovering bridges.

Fingerprints. A second piece of data extremely valuable for our work was the availability of the bridge sanitized fingerprint. It allowed us to link information from different CollecTor files and gather longitudinal information on individual bridges. Having a unique bridge identifier is fundamental to the granularity of CollecTor. Thus, it does not seem easy to remove them and instead raises the question of whether CollecTor should be a service only available to Tor maintainers. Another issue is that once the fingerprint of a public bridge is known, it can be hashed to find the specific bridge in CollecTor enabling near-real-time access to bridge's data (e.g., PTs offered, IP changes), as well as longitudinal statistics on the whole bridge lifetime. This could be addressed by adding a secret to

the fingerprint sanitization. However, this would prevent bridge owners to check statistics on their own bridges on CollecTor.

Usage Statistics. Rounding usage counters is a simple method to protect individual users while enabling statistics collection. Recent developments on privacy-preserving collection of statistics [35, 91] improve this protection and enable a wider range of statistics collection. Yet, these methods are not sufficient to prevent an adversary from ranking bridges according to their usage. We have shown that such rankings have two security implications. First, they enable an adversary to evaluate how successful is her blocking, not only at the global Tor Network level, but more worryingly for specific countries and PTs. Second, they allow an adversary to identify valuable (yet unblocked) bridges to target. Therefore, we believe that further research on privacy-preserving publishing of aggregated statistics is needed.

4.8.3 Security Implications of Bridge Properties

In this Section we discuss implications of the bridge properties we learned through the analysis of CollecTor data.

Bridge Stability. We found that public bridges can be coarsely classified in 55% volatile and 45% stable, where only stable bridges carry client traffic. Volatile bridges are short-lived and may be due to bridges installed on machines not always connected to the Internet, or by users testing how to run a bridge. Stable bridges are long-lived (median lifetime of 4 months) and rarely change IP address. Stability means once a user obtains the bridge information from BridgeDB it can use it for a long time. On the other hand, it also implies that current adversary blocking policies, e.g., the GFC removing blocks for bridge IP addresses every 25 hours [37], are extremely polite. Once an adversary finds a stable bridge, it can perform more aggressive blocking (up to months) or adaptive, by reconnecting to a bridge every day to check it is still active and the block should be renewed.

Use of Default Bridges. Our study of bridge importance reveals that default bridges carry over 90% of bridge users. Default bridges enable out-of-the-box use of Tor software, without the need to request bridges from BridgeDB. While censors may not be continuously blocking default bridges (otherwise they would not carry clients on censored countries), their massive usage enables easy disconnection of the bridges user base in response to events. Our measurements show that such blocking would disconnect nearly 90% of bridge clients in countries like Iran and Syria. Additionally, the fact that Tor users are educated to use the software out-of-the-box casts doubts about their ability to find alternative bridges when such blocking happens.

PT Deployment. We observe that 77% of public bridges only offer "vanilla Tor" and another 15% mix PTs with conflicting security properties (e.g., with and without reply protection), reducing the protection to that of the least safe transport. In general, the goal should be that bridges do not offer weak transports (e.g., vanilla) or deprecated ones (e.g., obfs2), but only PTs considered safe and without conflicting properties. The current PT deployment strategy in which bridge owners decide independently which PTs to offer from the complete pool of PTs is not optimal. There is a need for a faster way to remove PTs known to be unsafe (e.g., vanilla, obfs2). This could be achieved by adding automatic updates to Tor, enabling centralized decision on which PTs should no longer be offered and faster distribution of updates to disable them. In general, automatic updates would more quickly close the vulnerability window for any already fixed security issue. Since adding automatic updates may take time, the Tor software could in the mean time be configured to offer the strongest PT (e.g., obfs4) by default and to warn the user if two transports with conflicting security properties are about to be offered.

4.8.4 Security Implications of Uncovering Private Infrastructure

Discovery of private bridges and proxies is arguably more worrisome than de-anonymization of the IP address of public bridges. Their discovery allows an adversary to learn that IP addresses connecting to them correspond to Tor users that are members of the owner organization, and to use their IPs to geographically locate the users. This is particularly dangerous as private bridges and proxies may be run precisely by organizations trying to avoid such identification. One positive aspect is that since private bridges do not report to the Bridge Authority, nor are explored by any Tor-related service, it may be much easier to disable their OR port. This would prevent private bridges from appearing in scan search engines, and if they exclusively use strong PTs (e.g., obfs4), would thwart attempts of scanning to find them.

We have uncovered that multiple organizations are using private proxies as cheap replacements for private bridges. However, proxies are always in the same AS, which in 77% type I and II clusters is also the same AS hosting the backend OR. Once a proxy is discovered, it is possible to scan nearby IP addresses to find other proxies and the backend. In the case of backend relays, whose IP addresses are known, an adversary can perform localized scans around relay IP addresses to locate proxies. In those cases, using as backend a public bridge with a random OR port is preferable to a relay.

Proxies also have non-security implications for the Tor Network. First, they add an extra hop in addition to the 3-hop Tor circuit, which if the proxy lays across the Internet from the bridge increases the, already high, latency of Tor circuits. Second, proxies affect the usage statistics in CollecTor as connections from

multiple clients, potentially in different countries, are all counted as connections from the proxy IP address by the backend OR.

4.9 Related Work

The academic community has paid quite some attention to Tor bridges A first line of research related to this work deals with the proposal of PTs to avoid traffic analysis attacks [32, 76, 120, 128], and their detection [119]. The latter work studies the detectability of five popular PTs. Their results show that a determined adversary can reliably detect communications with bridges with a low false positive rate. We note that the quest for the best anti-censorship mechanism, where approaches that rely on raising the cost of blocking [52, 60, 128] seem most promising, is an orthogonal problem to the one studied in this paper where we are concerned with studying the security implications of currently deployed defenses.

A second line of work arises in response to censors using active probing to confirm that suspicious nodes are bridges [37]. This work also shows that the censors' systems operate in real time, are able to detect servers using five circumvention protocols and are regularly updated. Houmansadr et al. [51] show that PTs that mimic other protocols, e.g., [76, 120], are particularly sensitive to active probing since they only mimic part of the communication. Defenses against active probing are based on PTs that only reply upon being proven that the client knows a long-lived secret [5, 128] or a short-lived key [98].

A third relevant research line is dedicated to the discovery of bridges. McLachlan and Hopper [74] showed that it is possible to deanonymize bridge operators when bridges run in clients. In case bridges are dedicated servers, Ling et al [66] provide both active attacks where the adversary directly interacts with the bridge distributor, and passive attacks where she sets up a relay and enumerates bridges (i.e., non relay nodes) that connect to it. The work on Internet-wide scanning by Durumeric et al. [31] showed that the pattern in Tor certificates could be used to identify bridges, a property that we leverage to realize our measurement of the Tor bridge infrastructure. We show how to optimize enumeration by leveraging data in CollecTor and the availability of scan search engines to minimize investment in scanning infrastructure. We also present an alternative discovery technique based on non-Tor services bridges may also run, which is related to works leveraging leaks on scan search repositories to deanonymize hidden services [73].

Finally, Winter et al. [125] propose a tool to identify sybils in the Tor Network, i.e., relays owned by the same group, by studying configuration and uptime similarity. Our clustering has a similar goal as their nearest neighbor ranking and can be applied to relay descriptors as well, with the advantage that it does not require an input OR to compare with.

4.10 Ethical Considerations

Our measurements only leverage known limitations of the Tor Network, and only use leaks present in publicly accessible repositories such as Shodan, Censys or CollecTor. We purposefully avoid adding relays or bridges into the Tor Network, as well as exploiting any software vulnerability. We have no access to any traffic that is not ours, and hence we can not threaten the privacy of any Tor user. However, the data we collect contains the IP addresses and contact information of public and private bridges that must be kept private to preserve the security provided by the Tor Network. Thus, we do not disclose any bridge/proxy IP addresses, nor any personal information we may learn about its owners, but only provide aggregate data to illustrate important steps and findings.

This work has been approved by ethics review board of our institution, which has mandated that due to its sensitive nature the data must be protected with diligence, must not be disclosed to third parties, and must be deleted when the paper is published. We forwarded a copy of this draft with the Tor Project, that has already started taking measures [69] to prevent bridge targeting based on CollecTor public information.

4.11 Conclusion

In this work we provide the first systematic security analysis of the Tor bridge infrastructure. Our opportunistic measurements, made possible by taking advantage of two known Tor issues, allow us to discover thousands of bridges which we have used to gain understanding about the security properties of the bridge population. In particular we uncover the use of private proxy-based infrastructures likely to obtain IP diversity to access the Tor Network. We also study the impact on security of publicly available information such as that provided by CollecTor or scan search engines. Our results have implications for the Tor Project, since they indicate that the two issues we leveraged need to be solved as soon as possible, and that the information offered by CollecTor may need to be reduced; but also beyond, since we confirm that the information made available by public scan search engines should be taken into account when designing covert services.

Acknowledgments

The authors would like to thank the Tor Project their feedback. In particular, we are very thankful to Steven Murdoch for useful discussions about bridges' OR ports, and to Karsten Loesing for his feedback and for promptly launching a fix to CollecTor for anonymizing OR ports. Finally we would like to thank John Matherly and the Shodan project for their helpfulness.

This research was partially supported by the Regional Government of Madrid through the N-GREENS Software-CM S2013/ICE-2731 project, by the Spanish Government through the DEDETIS Grant TIN2015-7013-R, and by the European Commission through the H2020-ICT-2015 NEXTLEAP project (GA 688722). All opinions, findings and conclusions, or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the sponsors.

5

Conclusion and Future Directions

In the last years we have witnessed an ongoing interest from users towards Privacy Enhancing Technologies, and more specifically in the field of anonymity networks. Anonymity networks are the state of the art solution many users daily leverage to protect their personal information and communications and to access contents from countries where censorship is performed. An important component of an anonymity network is the hidden server, a machine whose IP address is kept secret. In this thesis we propose novel approaches and techniques for deanonymizing hidden servers. Our goals are both to raise awareness about security issues that might affect users of the anonymity network and owners of anonymous services, and to develop tools that can be used for building more secure anonymity networks.

In the first part of the thesis, we propose a novel approach for recovering the IP address of servers hosting hidden services in the Tor Network. To this end, we design, implement, and evaluate a tool that automatically explores a hidden service detecting location leaks. Those location leaks are leveraged for trying to unveil the IP address of the machine hosting the hidden service. Our tool, CARONTE, operates in an open-world model. It generates a list of candidate Internet endpoints that could be hosting the hidden hidden service, using the content and the configuration obtained from the hidden service. As a next step CARONTE connects to each candidate and compares the content downloaded from it with the one retrieved from the hidden service, in order to verify that the candidate Internet endpoint hosts the hidden service. We demonstrate our tool by running it on real hidden services and successfully deanonymizing over 100 of them. Both privacy-concerned owners of hidden services and law-enforcement agencies could benefit from such a tool. Owners of services that provide sensitive content (e.g., material from whistleblowers, or political dissidents) could use CARONTE to audit their service before making it available to the public. Law-enforcement agencies could apply CARONTE for trying to automatically identify addresses of hidden services hosting illegal content. Future work includes both ex-

tending CARONTE to support other location leaks (e.g., metadata from pictures) and other anonymity networks (e.g., I2P), as well as improving the coverage by extending CARONTE's crawling to all the content hosted on the hidden service, rather than just the main page.

In the second part of the thesis we perform the first systematic study of the Tor bridge infrastructure. Our study covers both the public bridge infrastructure available to all Tor users, and the previously unreported private bridge infrastructure, comprising private nodes for the exclusive use of those who know about their existence. To analyze the public bridge infrastructure we use public data accessible from the CollecTor service. Despite CollecTor sanitizing all sensitive information prior to its publication, we identify elements in the published data that may be detrimental for the security of public bridges. Using CollecTor we measured security relevant properties of public bridges. This measurement shows how the public bridges with clients are stable and they rarely change their IP address. As a direct consequence of this property, governments that restrict users' access to the anonymity network might switch to more aggressive strategies. We also show how additional services running on bridges can be abused for keeping track of IP changes. Furthermore we show that the presence of multiple transport protocols with conflicting security properties harms bridge security. After sharing our work with the Tor Project, the maintainers of CollecTor started sanitizing the OR port on which the bridge is listening for incoming connections

To study the private bridge infrastructure, we use an approach that leverages both scan search engines and two known Tor issues. During this process we discover 694 private bridges and 645 proxies that forward traffic to backend bridges or relays. Our results have implications for the Tor Project, since they indicate that the two issues we leveraged to deanonymize bridges need to be solved as soon as possible. We also show how the availability of scan search engines, which index data collected from Internet-wide scans, greatly lowers the cost of our deanonymization attack in terms of scanning infrastructure. Thus, the security of systems that require hiding components, e.g., Tor bridges, should be designed with this threat in mind. Finally we uncover the use of private proxy-based infrastructures likely to obtain IP diversity to access the Tor Network. Our future work includes evaluating possible solutions for the protocol-dependent issues used to deanonymize bridges, and developing strategies that would allow publication of network information through privacy-preserving statistics.

Anonymity networks have become an essential technology for many users that access content and communicate over the Internet. In this thesis we have shown how despite continuous progress in the development of more secure protocols, and without the need to introduce any malicious element in the anonymity network, still the privacy and security of their users can be seriously compromised. We have illustrated how focusing exclusively on the development of more sophisticated protocols and encryption schemes is not enough for protecting the anonymity of both users and service providers. We identify several sources of potential leaks

that can provide an attacker with valuable information about the location of a service or the identity of its owner. First of all, configuration issues when setting up the application to use the anonymity network. For example, in the scenario of Tor hidden services, this could be presence of files that were not supposed to be publicly accessible. Second, third-party services, such as search engines. When a service is available both within the anonymity network and on the Internet, the content downloaded from the hidden service might have been indexed by a search engine. If the content contains unique features, such as an extremely specific title or email address, the search engine itself could be used for identifying candidates. Also dedicated search engines, which scan the address space looking for open ports or addresses that offer a particular protocol, can become a valuable tool in the hands of an attacker. Protocols often contain unique patterns or fingerprints that allow the attacker to link an IP address with a particular protocol (as in the case of vanilla Tor and TLS). When this does not apply to the protocol adopted by the anonymity network, additional services running on a hidden server could be using different protocols that instead have those features. Finally, also publicly accessible datasets that contain statistics about the network are also a potential source of leaks. Even when this data goes through a sanitization process it can still be used for harming the security of critical elements of the anonymity network. Despite our work focusing on Tor, the possible sources of leaks that we exploited for deanonymizing Tor hidden servers could be leveraged to achieve the same goal within other anonymity networks. For this reason we hope that this thesis can be a help guiding developers and users of new generation anonymity networks.

Acknowledgments

This research was partially supported by the Regional Government of Madrid through the N-GREENS Software-CM project S2013/ICE-2731, by the Spanish Government through the StrongSoft Grant TIN2012-39391-C04-01 and the DEDETIS Grant TIN2015-7013-R, and by the European Commission through the H2020-ICT-2015 NEXTLEAP project (GA 688722).

Bibliography

[1] Tor Hidden Service (.onion) search. https://ahmia.fi/search/.

[2] Masoud Akhoondi, Curtis Yu, and Harsha V. Madhyastha. LASTor: A Low-Latency AS-Aware Tor Client. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2012.

[3] Ross Anderson et al. The eternity service. In *Proceedings of PRAGOCRYPT*, volume 96, pages 242–252, 1996.

[4] Andres Riancho. w3af. https://github.com/andresriancho/w3af.

[5] Yawning Angel. obfs4 (The obfourscator). https://gitweb.torproject.org/user/phw/scramblesuit.git/tree/doc/scramblesuit-spec.txt, 2014.

[6] Jacob Appelbaum. NSA targets the privacy-conscious. http://daserste.ndr.de/panorama/aktuell/nsa230_page-1.html.

[7] asn. Obfsbridges should be able to "disable" their ORPort. https://trac.torproject.org/projects/tor/ticket/7349.

[8] asn. GFW probes based on Tor's SSL cipher list, 2011. https://trac.torproject.org/projects/tor/ticket/4744.

[9] asn. On recent and upcoming developments in Pluggable Transports, 2014. https://blog.torproject.org/blog/recent-and-upcoming-developments-pluggable-transports.

[10] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-resource Routing Attacks Against Tor. In *Proceedings of the ACM Workshop on Privacy in Electronic Society*, 2007.

[11] Bernardo Damele A. G. and Miroslav Stampar. sqlmap. https://github.com/sqlmapproject/sqlmap.

[12] Alex Biryukov, Ivan Pustogarov, Fabrice Thill, and Ralf-Philipp Weinmann. Content and Popularity Analysis of Tor Hidden Services. In *Proceedings of the First International Workshop on Big Data Analytics for Security*, June 2014.

[13] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2013.

[14] Reporters Without Borders. Enemies of the Internet 2014: Entities at the heart of censorship and surveillance. http://12mars.rsf.org/2014-en/.

[15] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of Service or Denial of Security? In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007.

[16] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of Service or Denial of Security? - How Attacks on Reliability can Compromise Anonymity. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007.

[17] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. In *Communications of the ACM 24*, 1981.

[18] Chris Sullo. nikto. https://github.com/sullo/nikto.

[19] Christian Martorella. theHArvester. https://github.com/laramies/theHarvester.

[20] Nicolas Christin. Traveling the Silk Road: A Measurement Analysis of a Large Anonymous Online Marketplace. In *Proceedings of the 22nd International Conference on World Wide Web*, 2013.

[21] Nicolas Christin. Traveling the Silk Road: A measurement analysis of a large anonymous online marketplace. In *Proceedings of the 22nd international conference on World Wide Web*, pages 213–224. International World Wide Web Conferences Steering Committee, 2013.

[22] Adrian Crenshaw. Darknets and hidden servers: Identifying the true IP/network identity of I2P service hosts, 2011. Black Hat DC.

[23] Deep Web Links: .onion hidden service urls list. http://deepweblinks.org/.

[24] Roger Dingledine. Tor Project: Tor Rendezvous Specification. https://gitweb.torproject.org/torspec.git/tree/rend-spec.txt.

[25] Roger Dingledine. Tor Hidden Services, 2005. https://www.freehaven.net/~arma/wth3.pdf.

[26] Roger Dingledine and Jacob Appelbaum. How governments have tried to block Tor. In *28th Chaos Communication Congress*, 2012.

[27] Roger Dingledine and Nick Mathewson. Tor protocol specification. https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt.

[28] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, 2004.

[29] DuckDuckGo. http://3g2upl4pq6kufc4m.onion.

[30] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. A Search Engine Backed by Internet-Wide Scanning. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM SIGSAC Conference on Computer and Communications Security*, pages 542–553. ACM, 2015.

[31] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. ZMap: Fast Internet-wide Scanning and Its Security Applications. In Samuel T. King, editor, *USENIX Security Symposium*, pages 605–620. USENIX Association, 2013.

[32] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Protocol misidentification made easy with format-transforming encryption. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM Conference on Computer and Communications Security*, pages 61–72. ACM, 2013.

[33] Matthew Edman and Paul Syverson. AS-awareness in Tor Path Selection. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009.

[34] Tariq Elahi, Kevin Bauer, Mashael AlSabah, Roger Dingledine, and Ian Goldberg. Changing of the Guards: A Framework for Understanding and Improving Entry Guard Selection in Tor. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society*, 2012.

[35] Tariq Elahi, George Danezis, and Ian Goldberg. PrivEx: Private collection of traffic statistics for anonymous communication networks. In *ACM Conference on Computer and Communications Security*, 2014.

[36] Juan A. Elices, Fernando Perez-Gonzalez, and Carmela Troncoso. Fingerprinting Tor's Hidden Service Log Files Using a Timing Channel. In *Proceedings of the IEEE International Workshop on Information Forensics and Security*, 2011.

[37] Roya Ensafi, David Fifield, Philipp Winter, Nick Feamster, Nicholas Weaver, and Vern Paxson. Examining How the Great Firewall Discovers Hidden Circumvention Servers. In *ACM Internet Measurement Conference*, 2015.

[38] Nathan S. Evans, Roger Dingledine, and Christian Grothoff. A Practical Congestion Attack on Tor Using Long Paths. In *Proceedings of the 18th USENIX Security Symposium*, 2009.

[39] Facebook. 1 Million People use Facebook over Tor. https://www.facebook.com/notes/facebook-over-tor/1-million-people-use-facebook-over-tor/865624066877648/.

[40] Nick Feamster and Roger Dingledine. Location Diversity in Anonymity Networks. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society*, 2004.

[41] David Fifield. Meek. https://trac.torproject.org/projects/tor/wiki/doc/meek.

[42] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. Blocking-resistant communication through domain fronting. In *PoPETs*, 2015.

[43] Fredzupy. Ethiopia Introduces Deep Packet Inspection | The Tor Blog. https://trac.torproject.org/projects/tor/ticket/6140.

[44] David M Goldschlag, Michael G Reed, and Paul F Syverson. Hiding routing information. In *International Workshop on Information Hiding*, pages 137–150. Springer, 1996.

[45] Google. Certificate Transparency. http://www.certificate-transparency.org.

[46] Robert Graham. Reading the Silk Road configuration, October 2014. http://blog.erratasec.com/2014/10/reading-silk-road-configuration.html.

[47] Andy Greenberg. Wired: Global Web Crackdown Arrests 17, Seizes Hundreds Of Dark Net Domains, November 2014. http://www.wired.com/2014/11/operation-onymous-dark-web-arrests/.

[48] Glenn Greenwald and Ewen MacAskill. Boundless Informant: the NSA's secret tool to track global surveillance data, 2013. urlhttps://www.theguardian.com/world/2013/jun/08/nsa-boundless-informant-global-datamining.

[49] Nicholas Hopper. Short paper: Challenges in Protecting Tor Hidden Services from Botnet Abuse. In *Proceedings of the 18th International Conference on Financial Cryptography and Data Security*, 2014.

[50] Nicholas Hopper, Eugene Y. Vasserman, and Eric Chan-TIN. How Much Anonymity Does Network Latency Leak? *ACM Transactions on Information Systems Security*, 13(2):1–28, February 2010.

[51] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. The Parrot Is Dead: Observing Unobservable Network Communications. In *IEEE Symposium on Security and Privacy*, pages 65–79. IEEE Computer Society, 2013.

[52] Amir Houmansadr, Edmund L. Wong, and Vitaly Shmatikov. No Direction Home: The True Cost of Routing Around Decoys. In *Network and Distributed System Security Symposium*, 2014.

[53] Freedom House. Freedom on the Net 2015, 2015. https://freedomhouse.org/report/freedom-net/freedom-net-2015.

[54] Html::Similarity. http://search.cpan.org/~xern/HTML-Similarity-0.2.0/lib/HTML/Similarity.pm/.

[55] The Invisible Internet Project. https://geti2p.net/.

[56] International Computer Science Institute. The ICSI Certificate Notary. http://notary.icsi.berkeley.edu/.

[57] Rob Jansen and Nicholas Hopper. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In *Proceedings of the 19th Annual Network and Distributed System Security Symposium*, February 2012.

[58] Rob Jansen, Florian Tschorsch, Aaron Johnson, and Björn Scheuermann. The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium*, 2014.

[59] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2013.

[60] Josh Karlin and Daniel Ellard and Alden W. Jackson and Christine E. Jones and Greg Lauer and David Mankins and W. Timothy Strayer. Decoy Routing: Toward Unblockable Internet Communication. In *USENIX Workshop on Free and Open Communications on the Internet*, 2011.

[61] T. Ristenpart K. Dyer, S. Coull and T. Shrimpto. Protocol Misidentication Made Easy with Format-Transforming Encryption. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2012.

[62] Kaffeine. Guess who is back again, Cryptowall, January 2015. http://malware.dontneedcoffee.com/2015/01/guess-whos-back-again-cryptowall-30.html.

[63] Bryan Krebs. Silk Road Lawyers Poke Holes in FBI's Story, October 2014. http://krebsonsecurity.com/2014/10/silk-road-lawyers-poke-holes-in-fbis-story/.

[64] Rapid7 Labs. Rapid 7: Sonar Project. https://scans.io/study/sonar.ssl.

[65] Andrew Lewman. Update on Internet censorship in Iran. https://blog.torproject.org/blog/update-internet-censorship-iran.

[66] Zhen Ling, Junzhou Luo, Wei Yu, Ming Yang, and Xinwen Fu. Extensive analysis and large-scale empirical evaluation of tor bridge discovery. In Albert G. Greenberg and Kazem Sohraby, editors, *IEEE INFOCOM*, pages 2381–2389. IEEE, 2012.

[67] D. H. Lipman. Mailing list post: [tor-talk] vwfws4obovm2cydl.onion?, June 2012. https://lists.torproject.org/pipermail/tor-talk/2012-June/024565.html.

[68] Karsten Loesing. Evaluate, possibly revise, and then implement ideas for TLS certificate normalization. https://trac.torproject.org/projects/tor/ticket/7145.

[69] Karsten Loesing. Sanitize TCP ports in bridge descriptors. https://trac.torproject.org/projects/tor/ticket/19317.

[70] Karsten Loesing. [tor-dev] Introducing CollecTor (was: Spinning off Directory Archive from Metrics Portal). https://lists.torproject.org/pipermail/tor-dev/2014-June/006942.html.

[71] Karsten Loesing and Nick Mathewson. BridgeDB specification. https://gitweb.torproject.org/torspec.git/tree/bridgedb-spec.txt.

[72] John Matherly. Shodan, 2013. https://www.shodan.io/.

[73] Srdjan Matic, Platon Kotzias, and Juan Caballero. Caronte: Detecting Location Leaks for Deanonymizing Tor Hidden Services. In *ACM Conference on Computer and Communications Security*, 2015.

[74] Jon McLachlan and Nicholas Hopper. On the risks of serving whenever you surf: vulnerabilities in Tor's blocking resistance design. In Ehab Al-Shaer and Stefano Paraboschi, editors, *ACM Workshop on Privacy in the Electronic Society*, pages 31–40. ACM, 2009.

[75] Prateek Mittal, Ahmed Khurshid, Joshua Juen, Matthew Caesar, and Nikita Borisov. Stealthy Traffic Analysis of Low-latency Anonymous Communication Using Throughput Fingerprinting. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, 2011.

[76] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. SkypeMorph: protocol obfuscation for Tor bridges. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM Conference on Computer and Communications Security*, pages 97–108. ACM, 2012.

[77] MOTHERBOARD. Confirmed: Carnegie Mellon University Attacked Tor, Was Subpoenaed By Feds, 2016. urlhttp://motherboard.vice.com/read/carnegie-mellon-university-attacked-tor-was-subpoenaed-by-feds.

[78] Mozilla. Public Suffix List. http://publicsuffix.org/.

[79] mrphs. Guide to run an obfs4 bridge. https://trac.torproject.org/projects/tor/wiki/doc/PluggableTransports/obfs4proxy.

[80] Steven J. Murdoch. Hot or Not: Revealing Hidden Services by Their Clock Skew. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, 2006.

[81] Steven J. Murdoch and George Danezis. Low-Cost Traffic Analysis of Tor. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2005.

[82] Steven J. Murdoch and Piotr Zieliński. Sampled Traffic Analysis by Internet-exchange-level Adversaries. In *Proceedings of the 7th International Conference on Privacy Enhancing Technologies*, 2007.

[83] Netcraft. Keys left unchanged in many Heartbleed replacement certificates!, April 2014. http://news.netcraft.com/archives/2014/05/09/keys-left-unchanged-in-many-heartbleed-replacement-certificates.html.

[84] OONI Censorship Wiki: Iran. https://trac.torproject.org/projects/tor/wiki/doc/OONI/censorshipwiki/CensorshipByCountry/Iran.

[85] OONI Censorship Wiki: The Philippines. https://trac.torproject.org/projects/tor/wiki/doc/OONI/censorshipwiki/CensorshipByCountry/The_Philippines.

[86] Lasse Øverlier and Paul Syverson. Locating Hidden Servers. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2006.

[87] Lasse Øverlier and Paul Syverson. Valet Services: Improving Hidden Servers with a Personal Touch. In *Proceedings of the 6th International Conference on Privacy Enhancing Technologies*, 2006.

[88] The Tor Project. Obfs3. https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/tree/doc/obfs3/obfs3-protocol-spec.txt.

[89] The Tor Project. Tor Manual. https://www.torproject.org/docs/tor-manual.html.en.

[90] Michael G Reed, Paul F Syverson, and David M Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected areas in Communications*, 16(4):482–494, 1998.

[91] Rob Jansen and Aaron Johnson. Safely Measuring Tor. In *ACM Conference on Computer and Communications Security*, 2016.

[92] Roger Dingledine. Tor Project: Facebook, hidden services, and https certs. https://blog.torproject.org/blog/facebook-hidden-services-and-https-certs.

[93] runa. Kazakhstan uses DPI to block Tor. https://trac.torproject.org/projects/tor/ticket/6140.

[94] SameId. http://sameid.net/.

[95] Sarah Jamie Lewis. OnionScan. https://github.com/s-rah/onionscan, 2016.

[96] Bilal Shebaro, Fernando Perez-Gonzalez, and Jedidiah R. Crandall. Leaving Timing-channel Fingerprints in Hidden Service Log Files. *Digital Investigations*, 7:104–113, August 2010.

[97] sitting-duck. DirBuster. https://sourceforge.net/projects/dirbuster/.

[98] Rob Smits, Divam Jain, Sarah Pidcock, Ian Goldberg, and Urs Hengartner. BridgeSPA: improving Tor bridges with single packet authorization. In Yan Chen and Jaideep Vaidya, editors, *ACM Workshop on Privacy in the Electronic Society*, pages 93–102. ACM, 2011.

[99] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an Analysis of Onion Routing Security. In *Designing Privacy Enhancing Technologies: Workshop on Design Issue in Anonymity and Unobservability*, July 2000.

[100] The Guardian. NSA's PRISM surveillance program: how it works and what it can do, 2013. http://www.theguardian.com/world/2013/jun/08/nsa-prism-server-collection-facebook-google.

[101] The Guardian. Peeling back the layers of Tor with EgotisticalGiraffe, 2013. http://www.theguardian.com/world/interactive/2013/oct/04/egotistical-giraffe-nsa-tor-document.

[102] The Guardian. Tor: 'The king of high-secure, low-latency anonymity', 2013. http://www.theguardian.com/world/interactive/2013/oct/04/tor-high-secure-internet-anonymity.

[103] The Tor Project. censorship circumvention. https://blog.torproject.org/category/tags/censorship-circumvention.

[104] The Tor Project. CollecTor. https://collector.torproject.org/.

[105] The Tor Project. Ethical Tor Research: Guidelines. https://blog.torproject.org/blog/ethical-tor-research-guidelines.

[106] The Tor Project. Tor Cloud: Tor bridges in the Amazon cloud. https://cloud.torproject.org/.

[107] The Tor Project. Tor directory protocol, version 3. https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt.

[108] The Tor Project. Tor: Pluggable transports. https://www.torproject.org/docs/pluggable-transports.html.en.

[109] The Tor Project. Thoughts and Concerns about Operation Onymous, November 2014. https://blog.torproject.org/blog/thoughts-and-concerns-about-operation-onymous.

[110] The Tor Project. Tor Metrics, 2016. https://metrics.torproject.org/.

[111] Hidden Wiki: Tor .onion urls directories. http://thehiddenwiki.org/.

[112] Tor Blog. Iran blocks Tor; Tor releases same-day fix. https://blog.torproject.org/blog/iran-blocks-tor-tor-releases-same-day-fix.

[113] TorProject. Tor Bridges. https://metrics.torproject.org/clients-data.html.

[114] Torproject. Tor Project: Configuring Hidden Services for Tor. https://www.torproject.org/docs/tor-hidden-service.html.en.

[115] Torproject. Who uses Tor?, 2014. https://www.torproject.org/about/torusers.html.en.

[116] Michael Carl Tschantz, Sadia Afroz, David Fifield, and Vern Paxon. SoK: Towards Grounding Censorship Circumvention in Empiriscism. In *IEEE Symposium on Security & Privacy*, 2016.

[117] Vlad Tsyrklevich. Internet-wide scanning for bridges. https://lists.torproject.org/pipermail/tor-dev/2014-December/007957.html.

[118] Laurent Vanbever, Oscar Li, Jennifer Rexford, and Prateek Mittal. Anonymity on QuickSand: Using BGP to Compromise Tor. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, 2014.

[119] Liang Wang, Kevin P. Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. Seeing through Network-Protocol Obfuscation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 57–69. ACM, 2015.

[120] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. StegoTorus: a camouflage proxy for the Tor anonymity system. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM Conference on Computer and Communications Security*, pages 109–120. ACM, 2012.

[121] Tim Wilde. Great Firewall Tor Probing Circa 09 DEC 2011. https://gist.github.com/da3c7a9af01d74cd7de7.

[122] Tim Wilde. Knock Knock Knockin' on Bridges' Doors, 2012. https://blog.torproject.org/blog/knock-knock-knockin-bridges-doors.

[123] Philip Winter. Censorship by country: The Philippines. https://trac.torproject.org/projects/tor/wiki/doc/OONI/censorshipwiki/CensorshipByCountry/The_Philippines.

[124] Philipp Winter. ScrambleSuit Protocol Specification, 2013. https://gitweb.torproject.org/user/phw/scramblesuit.git/tree/doc/scramblesuit-spec.txt.

[125] Philipp Winter, Roya Ensafi, Karsten Loesing, and Nick Feamster. Identifying and characterizing Sybils in the Tor network. In *USENIX Security Symposium*, 2016.

[126] Philipp Winter and Stefan Lindskog. How the Great Firewall of China is Blocking Tor. In *FOCI*, 2012.

[127] Philipp Winter and Stefan Lindskog. How the Great Firewall of China is Blocking Tor. In *USENIX Free and Open Communications on the Internet*, 2012.

[128] Philipp Winter, Tobias Pulls, and Jürgen Fuß. ScrambleSuit: a polymorphic network protocol to circumvent censorship. In Ahmad-Reza Sadeghi and Sara Foresti, editors, *ACM Workshop on Privacy in the Electronic Society*, pages 213–224. ACM, 2013.

[129] Wired. A simple guide to GCHQ's internet surveillance programme Tempora, 2013. http://www.wired.co.uk/news/archive/2013-06/24/gchq-tempora-101.

[130] Matthew Wright, Micah Adler, Brian N. Levine, and Clay Shields. Defending Anonymous Communications Against Passive Logging Attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2003.

[131] Yawning. Obfsproxy4. https://github.com/Yawning/obfs4/blob/master/doc/obfs4-spec.txt.

[132] Sebastian Zander and Steven J. Murdoch. An Improved Clock-skew Measurement Technique for Revealing Hidden Services. In *Proceedings of the 17th USENIX Security Symposium*, 2008.