UNIVERSITEIT VAN AMSTERDAM

System and Network Engineering

MSc Security and Network Engineering
Research Project I

# Invisible Internet Project (I2P)

February 10, 2019

Tim de Boer
tim.deboer@os3.nl

Vincent Breider
vincent.breider@os3.nl

*Assessor*
Prof. dr. ir. C.T.A.M. de Laat
University of Amsterdam

*Supervisor*
Fons Mijnen
Deloitte

**Abstract**

The Invisible Internet Project (I2P) is a decentralised message oriented mixnet, that allows users to communicate in an anonymous and encrypted manner. Invisible Internet Project (I2P) provides an excellent opportunity for people requiring anonymity to bypass oppressive regimes or censorship. However, it also may provide problematic in corporate environments where I2P-routers over-utilise limited network resources or provide a way for malware to exfiltrate data. In this paper, we will investigate the possibility for an IDS to detect I2P network traffic and positively identify peers. We provide a background on the inner workings of I2P, and we demonstrate how to identify I2P routers during the bootstrapping phase, and show how statistical traffic analysis reveals a pattern in the message length that may aid in the fingerprinting of I2P routers that have been participating in the network for a while.

# Contents

# 1 Introduction

There are multiple anonymity networks (e.g. The Invisible Internet Project (I2P), The Tor Project (Tor) and FreeNet) that intend to protect the end-user from any tracking, surveillance and monitoring by third parties such as Internet Service Provider (ISP), network administrators and governments[1].

I2P is an anonymous, self-organising and decentral overlay network[2], which relies on an extension of the Onion routing technique named Garlic routing[3]. Garlic routing, a variant of the Onion routing protocol (used by Tor), is a technique for building paths, or tunnels, through a series of peers, and then using that tunnel. The originator repeatedly encrypts messages which are then decrypted by each hop as it passes through a tunnel. During the building phase, only the routing instructions for the next hop are exposed to each peer. The critical difference is that with garlic routing a router waits for other relayed messages and packs them together in an encrypted bundle along with its message while in onion routing this is not the case, making statistical analysis with the purpose of deanonymization harder.

# 2 Research Question

With this research, we aim to determine if it is possible for an entity that has full control of network traffic to identify and fingerprint peers that are participating in the I2P network. When reading the developer documentation [4], we can see that - to the developers - anonymity is defined not as to make it difficult to identify who is hosting an I2P router at an Internet Protocol (IP)-address, but rather to make it difficult in identifying who send what data at what time to whom for what purpose within the I2P network. Unfortunately, there are real-world scenario's where merely knowing that a router participates in an anonymisation network is enough to undertake actions that will prosecute or censor citizens. As is the case in some countries where the use of encryption for any application is restricted or prohibited[5], or using software that is not sanctioned by the government can get people prosecuted or have their internet censored (e.g. the Great Firewall of China [6]).

However, there are less nefarious motivations to identify I2P router traffic, such as that of a security operations centre of a large organisation, which needs to identify undesired network communications originating from its local network. To discontinue unauthorised bandwidth utilisation, or for example to identify malware. Recently, malwares known as I2Ninja, Dyre and CryptoWall 3.0 have been identified that use I2P as a covert channel to communicate with a command and control centre[7]. Our research, therefore, focuses on the following question: Is it possible for an entity that intercepts network traffic to fingerprint and positively identifies hosts that are participating in the I2P network?

To support the main research question above the following sub-questions are defined:

- How does the I2P network operate, how does the protocol work?
- Is it possible for a traditional IDS to identify I2P traffic during the router initiation phase?
- Can traffic be identified by scraping the distributed hash table (NetDB) for IP addresses of known participants?

- Can the I2P protocol be fingerprinted using statistical traffic analysis?

# 3 Related Work

Bazli et al. investigated how forensic investigation into the I2P network could be conducted, by examining the forensic artefacts of the I2P installer. Bazli et al. describe techniques that allow tracking the user's behaviour within the I2P network by comparing the address book of against a reference database, taking over an address book registrar, locating an I2P node by network performance and identifying the behaviour of new I2P users by creating a false mirror site of existing eepsites[8].

Timpanaro et al. performed a study in which they design a distributed monitoring system for the I2P network. By deploying many floodfill routers in the network that retrieve and replicate the NetDB Distributed Hash Table (DHT), it is possible to build a dataset of leasesets that can be queried to determine the Web and Filesharing applications running on I2P at certain routers. Timpanaro et al. show this to be able to chart the use of the I2P network[9].

Hjelmvik and John looked closer into network protocol obfuscation and how statistical analysis can be used to identify a protocol despite the obfuscation. They created the Statistical Protocol IDentification (SPID) framework to identify protocols using statistical analysis of application data and conclude that statistical analysis is of great help in the identification of network protocols where static patterns based on fingerprints fail[10].

# 4 The Invisible Internet Protocol (I2P)

I2P has implemented its a communication network and protocol stack on top of Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). I2P ensures that participants can send and receive messages with other participants securely and anonymously. Other software such as Tor or Freenet provides similar functionality. However, I2P differs from Tor and Freenet as Freenet is an anonymous distributed data storage which has services build on top that allows for site browsing or message-boards. I2P, like TOR, uses a variant of onion routing named garlic routing to create anonymous connections. Tor aims at connecting to clearnet clients, where I2P is focused on allowing clients to anonymously access services internal to the I2P network, such as Bit-Torrent clients or websites hosted within the I2P network known as eepsites [2]. Besides that, I2P is distributed whereas Tor has directory services that store statistics and information of Tor-nodes in a central place. I2P uses the network database (NetDB) to store information on the I2P network.

The NetDB is implemented as a DHT and is propagated via so-called floodfill routers using the Kademlia algorithm, making I2P a decentralised network. For a router to start participating in the I2P network it requires a part of the NetDB in which the information resides it needs to communicate to other participants of the network. Obtaining the NetDB is called bootstrapping and happens by 'reseeding' the router. By default, a router will reseed the first time it starts by querying some hard-coded domain names via the Domain Name System. When a router can successfully establish a connection to one of these domains a Transport Layer Security (TLS) connection is set up through which the router downloads a signed partial copy of the NetDB. Once the router can reach at least one other participant, the router will query for other parts of the NetDB it does not have itself. Alternatively reseeding can be done manually by uploading a zip-archive which has a part of the NetDB.

I2P differentiates between the addressing of routers (RouterInfos) and destinations (LeaseSets). Routers relay I2P messages to other routers, whereas the destinations offer services on top of the I2P network protocol stack and thus act as endpoints. These services include but are not limited to websites (eepsites), file-sharing services, or Internet Relay Chat (IRC) services. A router always relays messages and optionally can act as an endpoint if it publishes a LeaseSet in the NetDB for a service that it is hosting. Routers publish their RouterInfo directly into the NetDB, which consists of an IPv4 or IPv6 addresses, the operating TCP and UDP port, and a pair of public keys. LeaseSets cannot be directly published in the NetDB because this would affect anonymity. A LeaseSet consists of a Tunnel address through which messages can be sent to the service, an expiration time of the LeaseSet, and a public key to be able to encrypt messages destined for the endpoint. Because I2P is designed for destinations to remain anonymous, these LeaseSets have to be published anonymously into the NetDB, so that correlating a LeaseSet with the RouterInfo of the hosting router becomes very difficult [4].

To transmit messages anonymously (e.g. to publish a LeaseSet) a client creates a pool of Inbound and Outbound Tunnels, these Tunnels expire every 10 minutes. A Tunnel is a series of routers, selected from the NetDB, that forward messages from or towards a client using a variant of onion routing. Each hop in this tunnel can have a different role, either being a Gateway, a Participant or an Endpoint. An I2P router has two types of Tunnels, Client Tunnels, and Exploratory Tunnels. Client Tunnels are used for anonymously offering LeaseSets to the NetDB and accessing destinations within the I2P network, and to offer and connect to endpoints. Exploratory Tunnels are used for peer selection, to identify other suitable candidates in the I2P network with whom the router will establish future Tunnels.

Tunnels in I2P are unidirectional, meaning that different Tunnels are used for transmitting and receiving messages. For instance, if Alice wants to communicate with Bob - who is hosting a service - she establishes a pool of Outbound and Inbound Tunnels. She does this by querying the NetDB for a set of peers and obtains their RouterInfo. Alice then establishes an encrypted connection with the first hop and sets up the first part of the Outbound Tunnel. Through this first hop, she will send the messages required to create Tunnels with the second and third router and completing the Outbound Tunnel as shown in figure 1. Alice repeats this until she has a set of Inbound and Outbound Tunnels called Tunnel Pools. Typically I2P Tunnels have a length of two hops, but can be configured to have at most five hops. To communicate with Bob Alice selects an Outbound Tunnel to send a message through. In this Tunnel the first hop is a Gateway, then the intermediate routers act as Participants in the Tunnel. Finally, the Endpoint of the Tunnel transmits the message via intermediate routers from Alice to the Inbound Tunnel of Bob, as shown in figure 2. To be able to receive messages Bob will publish the LeaseSet of the service he is hosting through an Outbound Tunnel into the NetDB, because it is costly to look up Tunnels in the NetDB Alice will send Bob the information needed to respond to her messages using one of her active Inbound Tunnels [4]. To prevent routers that are part of a Tunnel from snooping inside the messages they are relaying, a form of Onion Routing is used called Garlic Routing. With Onion Routing transmitted messages are encrypted multiple times (often referenced as the layers of an onion) and can be thought of

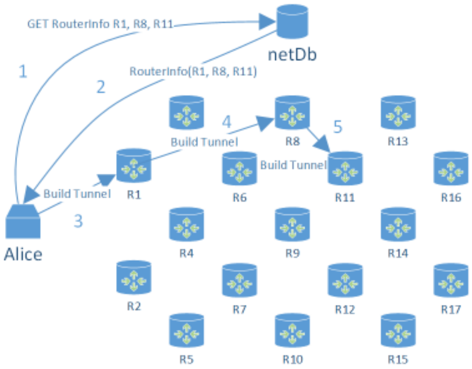


Figure 1: Creation of an Outbound Tunnel, by querying RouterInfo from the NetDB.

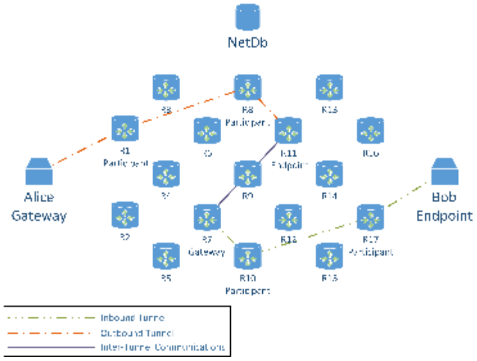


Figure 2: Typical communication of messages in the I2P network using inbound and outbound tunnels.

like an envelope with a seal (the act of encrypting a payload) containing another sealed envelope. Each 'envelope' consists of a payload encrypted with a router's public key and routing instructions to the next hop in the Tunnel, in turn each payload also consists of such an envelope, this repeats itself for every hop, and each hop in the tunnel can therefore only operate on the message data that is meant for that hop, it also means that a hop cannot see where a message is going beyond the next hop. In the 'envelope' of the last hop of a Tunnel is a message encrypted with the public key of the destination, this is again encrypted by the Inbound Tunnel in multiple envelopes to be transmitted further along the Inbound Tunnel towards the destination. Garlic Routing differs from Onion Routing because it can pack

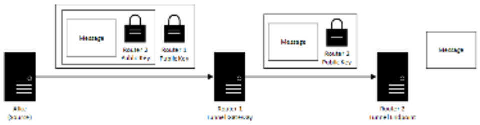


Figure 3: An illustration of layered encryption used in onion routing.

multiple messages together as 'garlic cloves' to form a 'garlic' message, which is then further routed to the next hop, as depicted in figure 4. Garlic Routing gives

I2P the advantage that timing-based traffic analysis is difficult to achieve. For instance, if Alice sends a message using Onion Routing to Bob and an adversary named Eve is eavesdropping on the network connections of Alice and Bob she may infer from the timing that Bob receives a message several seconds after Alice had sent it [11]. When Alice sends a message using Garlic Routing, she bundles the message she wants to send with other messages she received from participating in other Tunnels that share the same Tunnel Endpoint, when the garlic reaches a hop where the gloves have different destinations it is split again and merged into other garlic messages. Eve now sees that Alice has sent a message, but a few seconds later Eve cannot be sure if Bob received the same message because several other participants also received a message [4]. More specifically Eve cannot see if the message originated from Alice or if she is relaying it as she participates in other tunnels.

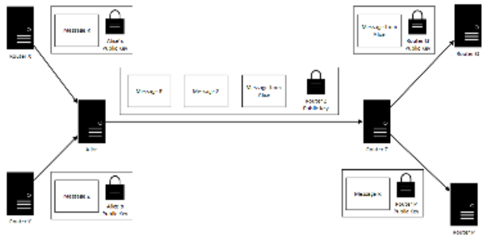


Figure 4: An illustration of messages packed as garlic cloves in garlic routing.

# 5 Approach

To be able to fingerprint the I2P protocol and positively identify hosts running I2P routers, we need to capture the network traffic in various circumstances and look for identifiable attributes. In this section, we describe the lab environment that we use to capture network traffic of I2P routers, and we describe the experiments that we perform to answer the research questions posed in section2.

## 5.1 Lab Environment

To conduct the experiments we need to capture the network traffic of routers that participate in the live I2P network as they are relaying messages. We can achieve this by creating an Infrastructure as Code using Ansible for configuration management and GitLab to store the code under version control [12]. Using an Infrastructure as Code allows us to provision Ubuntu-based virtual machines configured to run an I2P-router. The experiments run in the following environment:

- Hyper-visor running Ubuntu 18.04 with kernel 4.15-0.39-generic and xen-hypervisor-4.9-amd64

- Virtual Machines with Ubuntu 18.04 with kernel 4.15-0.43-generic, 10GB hard-disk, 1GB of Ram and two virtual CPU cores, running I2P-router software versions 0.9.37-0 and 0.9.38-0.

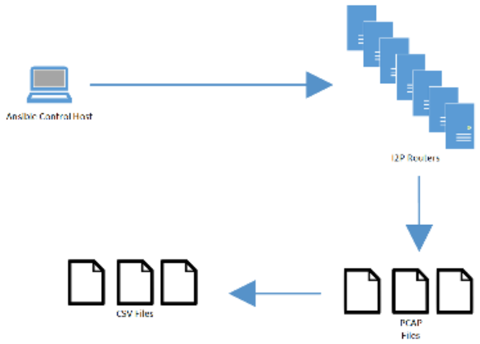


Figure 5: The lab environment generated with an Infrastructure as Code, Packet Capture (PCAP)s are converted to Comma-separated values (CSV) files using a Bash script. These CSV files are then processed further using Python and R.

### 5.2 Experiments

The first experiment will focus on the bootstrapping phase of the router. We deploy a single Virtual Machine (VM) with an I2P router installed and start a traffic capture using an Ansible task right before the I2P daemon starts. This approach ensures that we capture the network traffic when the router reseeds itself by querying the domain names that are hard-coded in the router software. As Domain Name System (DNS) and DNS are protocols that operate on well-known ports with identifiable parameters, the behaviour of the router during bootstrapping should be trivial to detect in the captured data.

For the second experiment, we will look at the router's network traffic when it has been running for 24 hours, meaning that it has reseeded and only will transmit TCP segments and UDP datagrams. We expect no identifiable parameters to be in this traffic and that the payloads are encrypted and padded. When comparing I2P with other protocols such as Tor, during a connection a typical TLS handshake takes place. This handshake sends a TLS Client Hello message of which the certificate's Common Name (CN) discloses that the traffic is the Tor protocol [13]. This example shows how network meta-data can provide information on the protocol despite it using encryption. As we expect no identifiable parameters we will use statistical traffic analysis to try and fingerprint I2P traffic and look at the following attributes:

- The use of TCP or UDP as the transport protocol.
- The top-talker port of the intercepted hosts.
- The port numbers that are used by the routers.
- Length in bytes of the transport layer Protocol Data Unit (PDU) payload.

To extract the attributes from the network PCAP files, we will convert them [14] extract the attributes mentioned above using Bash and Python, to visualise the data we will use R [15].

For our final experiment, we look from the perspective where we also operate an I2P router and are provided by updates of the NetDB from other floodfill routers. We hypothesise that the RouterInfo of every router must once be part of the NetDB in order to receive traffic from other participants. We will create a parser that extracts the IP-addresses of routers participating in the I2P-network and match these against the source IP-address our routers in the network traffic captures [16].

# 6 Results

This section contains the result from the experiments done using the setup as described in section 5.

When an I2P router needs to communicate with other routers, it has to look up the RouterInfo of its peers in the NetDB. After a router is installed and the application is started for the first time, it requires an initial copy of the NetDB with some entries. A newly installed I2P router does not have a NetDB and requires a preseed. This phase is also known as bootstrapping or initialisation phase. When the I2P router is bootstrapped, it can communicate with some other participating I2P router and will propagate the rest of the NetDB, which we call the operational phase. By using multiple I2P routers in our lab-environment, we were able to capture network traffic in a controlled manner and analyse the different phases.

## 6.1 Initialisation phase

During the bootstrapping phase, by default, the router will attempt to reseed (i.e. obtain an initial copy of the netDb) over Hypertext Transfer Protocol Secure (HTTPS) which results in the following, observable DNS queries, the domains of which are hard-coded in the application.

- reseed.i2p.net.in
- i2p.novg.net
- i2pseed.creativecowpat.net
- itoopie.atomike.ninja
- reseed.onion.im
- reseed.memcpy.io
- reseed.atomike.ninja
- i2p.mooo.com
- download.xxlspeed.com
- netdb.i2p2.no
- reseed.i2p-projekt.de

The router will attempt to download the *i2pseeds.su3*-file, which is a signed file containing around 80 router entries. This file is downloaded via HTTPS where a Certificate Authority (CA) could sign the certificate or is validated with the pre-bundled certificates. These certificates contain identifiable parameters, such as; fingerprint, common-name and dates.

Downloading the preseed file will generally fail for all listed reseed-servers, due to a check on the User-Agent. The Hypertext Transfer Protocol (HTTP) requests made by the I2P router is always made with the User-Agent string of *"Wget/1.11.4"*. After downloading, verifying and unpacking the initial seed, the router can populate the netDB further by querying other peers.

## 6.2 Operational phase

After around 20 minutes the I2P router is fully operational and has established some tunnels through which it is relaying Garlic Routed messages. In this phase, the router maintains a constant rate of throughput, and periodically (i.e. every 10 minutes) revokes and creates new tunnels. From a traffic capturing perspective, only TCP and UDP PDUs are transmitted of which the payload contains data that appears random.

### 6.2.1 Traffic analysis

The network captures we gained from our lab-environment contain almost solely I2P traffic. The first thing we have noticed is the fact that each I2P-router communicates over consistently over a randomly chosen port during the initialisation for TCP and UDP. Filtering the network capture data on the used I2P router port, which is known in our controlled lab-environment, resulted in a clean capture of I2P traffic.

As shown in figure 6 and figure 7 we can verify that these ports are chosen fairly random, and no single set of ports is constantly reused.

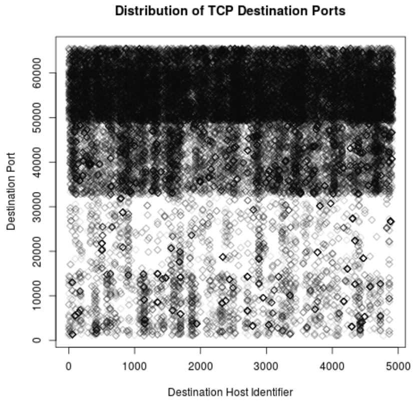


Figure 6: Distribution of TCP Destination Ports

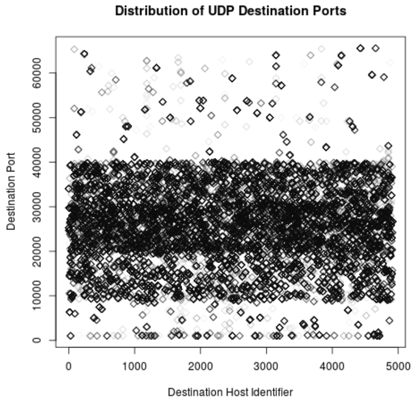
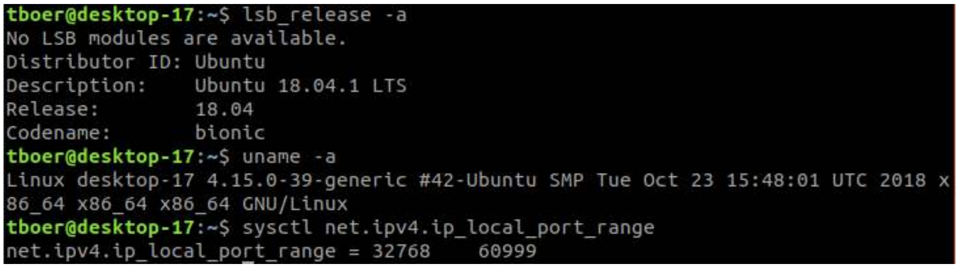


Figure 7: Distribution of UDP Destination Ports

Looking more closely on figure 6 we see a denser area which corresponds with the Dynamic Port Range (49152-65535) defined by Internet Assigned Numbers Authority (IANA)[17]. This range of ports is never assigned and mainly used for temporary and dynamic connections, such as Network Address Translation (NAT). A second denser area can be observer right under the IANA Dynamic Port Range which is caused by the deviant range (32768-61000) used by the Linux Kernel since version 2.6.22[18] release on 17 December 2003. This range is still used in modern Linux distributions, such as Canonical Ubuntu 18.04, with modern Kernel version, as seen in figure 8. Also in figure 7 we observed different bandwidths in the usage of UDP ports, starting from 10.000, with increments of 10.000, till 40.000. A logical explanation could be; different implementations of UDP Port forwarding ranges.

We expect I2P traffic to have an evenly spread distribution of the message length because messages are padded. Analysing the message length of I2P related traffic we observed that some message lengths are over-represented as shown in figure 9 and figure 10. In particular, for UDP messages with sizes in between 50 and 100 bytes and around 300, 500, 600 and 1100 bytes appear to be more common. For TCP messages of sizes in between 1000 and 1100 bytes, 1400 and 1500 and around 800 bytes appear to be more common. In the TCP data we can also see



```
tboer@desktop-17:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.1 LTS
Release:        18.04
Codename:       bionic
tboer@desktop-17:~$ uname -a
Linux desktop-17 4.15.0-39-generic #42-Ubuntu SMP Tue Oct 23 15:48:01 UTC 2018 x
86_64 x86_64 x86_64 GNU/Linux
tboer@desktop-17:~$ sysctl net.ipv4.ip_local_port_range
net.ipv4.ip_local_port_range = 32768    60999
```

Figure 8: Ephemeral ports used in modern Linux distribution

an over-representation of messages with length 0, these are empty TCP segments with the ACK flag set.

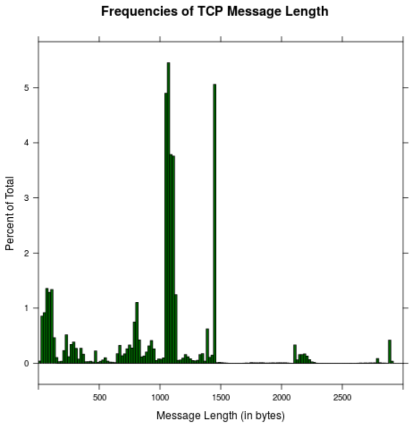


Figure 9: Frequencies of TCP message lengths.

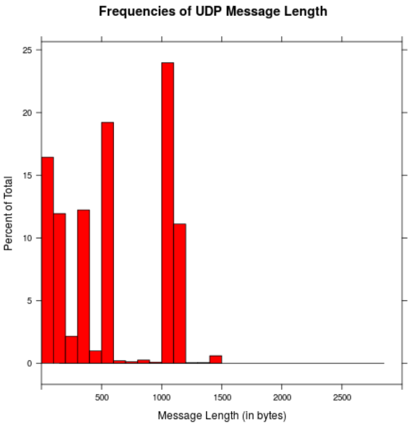


Figure 10: Frequencies of UDP message lengths.

Looking more closely, by using scatter-plots, we see message length of the used message length for TCP in figure 11 and UDP in figure 12. These deterministic patterns, the horizontal lines, could have a correlation with the number of participating I2P routers, i.e. the number of hops, in a tunnel.

### 6.2.2 Analysis on the NetDB

As described in section 4, the I2P router has a database containing meta-data on the I2P-network such as the RouterInfos of other I2P routers, the so-called NetDB. The database contains public-keys, IP-addresses, LeaseSets and ports among other things. Because it contains IP-addresses of participating routers it is interesting to harvest this data and compare it to live network traffic captures, to see if it is possible to identify I2P routers within a network segment.

For this purpose, we were able to build a parser for the NetDB database and compared the containing IP-address and port combinations with other NetDB databases. Interestingly these databases are not fully intersecting, as the databases contain different sets of IP-address and port combinations. Even geographically close I2P routers have of different sets. We observed that although six routers were running at the time, only two showed up in the NetDb. Presumably, the NetDB

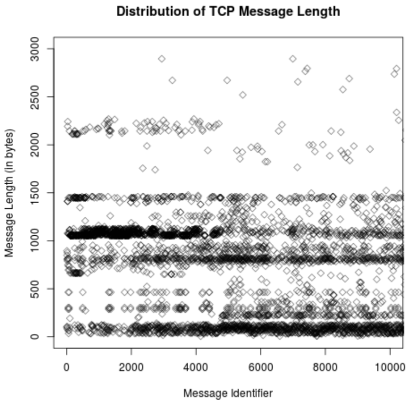


Figure 11: Distribution of message length with TCP traffic

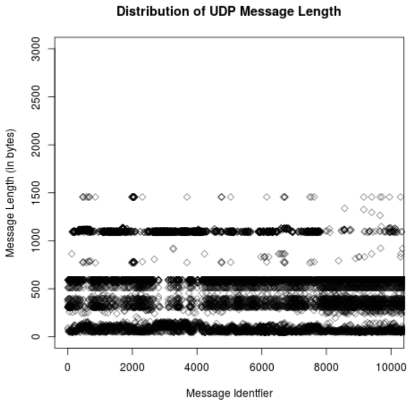


Figure 12: Distribution of message length with UDP traffic

databases contain a small set of I2P routers and routers will actively query for missing information as the browsing behaviour of the user requires it.

# 7 Conclusion

We can conclude from the experiments that it is possible to identify I2P traffic during the initialisation/bootstrapping phase as we could quickly detect the DNS queries and HTTPS requests towards the clearnet. Preventing I2P routers from bootstrapping in a controlled network can be done by blocking the DNS queries, blocking reseed related domains, blocking HTTPS traffic with TLS certificate Common Names and User-Agent HTTP headers by using an Intrusion Prevention System (IPS) or HTTP(S) proxy. Do note, however, that it is possible for an I2P router to bootstrap itself by letting the user manually upload the initial NetDB file that it obtained out-of-band thus bypassing detection.

After an I2P router reseeds, its network traffic shows no identifiable parameters (e.g. protocol headers or fields), making it difficult to detect with an Intrusion Detection System (IDS) traditionally. By performing a statistical analysis of the TCP and UDP payload data length, we demonstrate that it is possible to find a deterministic pattern. This pattern can potentially be used to identify I2P routers that have already bootstrapped. However, to give a definitive conclusion on how this pattern characterises I2P more research is needed. Statistical analysis on the source and destination ports shows a distribution from ports within the non-privileged range (i.e. higher than 1023) all the way up to the dynamically allocated port ranges. We conclude that the distribution of protocol source and destination ports is too widely dispersed to serve as an IDS metric practically. The data does show that I2P characterises itself as a protocol in which incoming and outgoing traffic is symmetrical and of high throughput, identifying the I2P routing port as 'top-talker' over time.

Parsing the NetDB to identify participating routers provides inconsistent results as not all routers in the experiment showed up in each others NetDB. We speculate that this the result of not using a broader set of routers that are more geographically dispersed. Determining if this method will yield more usable results is a topic of future research.

# 8 Discussion

The patterns in the message length are hard for an IDS to detect as most IDS systems look at discrete values in a single frame, such as the TCP port numbers, or a header that may be present in the application layer data. However, the pattern in the message data length reveals itself when observing I2P messages as traffic flows over time. Making it possible to identify I2P traffic as part of a forensic investigation in hindsight, but it is infeasible for an IDS or IPS to use as a metric as it often needs to act on this information in real-time.

An alternative approach would be to use the information in the NetDB to query the IP-address of participating routers and match this against the inspected traffic. However, the NetDB changes in such a rapid manner that it would require a constant update of the IDS detection ruleset.

# 9 Future work

I2P allows for different operational modes that aid in testing and debugging. In particular, routers can be forced to only find other routers by controlling where routers reseed from and which RouterInfos are present in the NetDB. In such a test network it is easier to eliminate variables such as Tunnel hop-length as this can be a fixed value among all network participants. To further investigate how the pattern in the data length characterises I2P traffic a follow-up study should compare our data. This study should compare our data with traffic captured from the private I2P network setup where Tunnel lengths are fixed on a single value, but also with traffic captured from other protocols that use Onion Routing, such as Tor, to see if the deterministic pattern is a result of the layered encryption each hops adds or strips from a Tunnel.

The experiments in this paper mainly focused on passive techniques to identify and fingerprint I2P routers, an interesting subject of research is how active probing techniques can identify I2P routers by eliciting identifiable behaviour. More specifically, since the reseeding of a router is trivial to detect, is it possible to exploit an I2P router and force it to reseed over the network?

Finally, it is interesting to see if it is possible to parse and accumulate the NetDB of geographically dispersed routers to build a complete dataset of the entire I2P network. This dataset can then be used to maintain a historical archive of routers that have appeared in the network database, and use this to create a reputation list to be used by an IDS.

# 10 Acknowledgements

We want to thank Fons Mijnenen and Vincent van Mieghem for their feedback and supervision.

We would also like to thank the OS3 Core-team for the feedback sessions and opportunities as well as the educational facilities that were at our disposal to perform this research.

In particular, we would like to thank Paola Grosso for her insight and feedback on our statistical analysis, and advice on data visualisation.

# References

[1] *The Invisible Internet Project.* 10 Jan. 2019. URL: https://geti2p.net/en/.

[2] *Intro - I2P.* 10 Jan. 2019. URL: https://geti2p.net/en/about/intro.

[3] *Garlic Routing - I2P.* 7 Jan. 2019. URL: https://geti2p.net/en/docs/how/garlic-routing.

[4] *I2P's Threat Mode.* 3 Feb. 2019. URL: https://geti2p.net/en/docs/how/tech-intro.

[5] Bert-Jaap Koops. *Crypto Law Survey.* 10 Feb. 2019. URL: http://cryptolaw.org/.

[6] Roya Ensafi et al. "Examining How the Great Firewall Discovers Hidden Circumvention Servers". In: *Internet Measurement Conference.* ACM, 2015.

[7] Etay Maor. *Out of the Shadows: i2Ninja Malware Exposed.* URL: https://securityintelligence.com/shadows-i2ninja-malware-exposed/ (visited on 20/11/2013).

[8] Behnam Bazli, Maxim Wilson and William Hurst. "The dark side of I2P, a forensic analysis case study". In: *Systems Science & Control Engineering* 5.1 (2017), pp. 278–286. URL: https://doi.org/10.1080/21642583.2017.1331770.

[9] Juan Pablo Timpanaro, Isabelle Chrisment and Olivier Festor. *Monitoring the I2P network.* English. Preprint. Oct. 2011. URL: http://hal.inria.fr/inria-00632259.

[10] Erik Hjelmvik and Wolfgang John. "Breaking and Improving Protocol Obfuscation". In: (Jan. 2010).

[11] *I2P's Threat Model.* 30 Jan. 2019. URL: https://geti2p.net/en/docs/how/threat-model.

[12] 8 Feb. 2019. URL: https://gitlab.os3.nl/tboer/rp1/tree/master/IaC.

[13] Srdjan Matic, Carmela Troncoso and Juan Caballero. "Dissecting Tor Bridges: A Security Evaluation of their Private and Public Infrastructures". In: *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017.* 2017. URL: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/dissecting-tor-bridges-security-evaluation-their-private-and-public-infrastructures/.

[14] 8 Feb. 2019. URL: https://gitlab.os3.nl/tboer/rp1/blob/master/pcap_analysis/script/pcap_to_csv.sh.

[15] 8 Feb. 2019. URL: https://gitlab.os3.nl/tboer/rp1/tree/master/script.

[16] 8 Feb. 2019. URL: https://gitlab.os3.nl/tboer/rp1/blob/master/pcap_analysis/netdb_parser.py.

[17] et al. M. Cotton. *Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry.* URL: https://tools.ietf.org/html/rfc6335 (visited on 07/02/2019).

[18] Mark Glines. *[TCP]: Use default 32768-61000 outgoing port range in all cases.* URL: https://github.com/torvalds/linux/commit/3f196eb519a419bf83ecc22753943fd0a0de4 (visited on 01/07/2007).

# A Acronyms

## Acronyms

**CA** Certificate Authority. 8

**CN** Common Name. 7

**CSV** Comma-separated values. 7

**DHT** Distributed Hash Table. 3

**DNS** Domain Name System. 7, 8, 11

**HTTP** Hypertext Transfer Protocol. 8, 11

**HTTPS** Hypertext Transfer Protocol Secure. 8, 11

**I2P** Invisible Internet Project. 1–12

**IANA** Internet Assigned Numbers Authority. 9

**IDS** Intrusion Detection System. 11, 12

**IP** Internet Protocol. 2, 4, 8, 10, 12

**IPS** Intrusion Prevention System. 11, 12

**IRC** Internet Relay Chat. 4

**ISP** Internet Service Provider. 2

**NAT** Network Address Translation. 9

**PCAP** Packet Capture. 7

**PDU** Protocol Data Unit. 7, 9

**SPID** Statistical Protocol IDentification. 3

**TCP** Transmission Control Protocol. 3, 4, 7, 9–12

**TLS** Transport Layer Security. 3, 7, 11

**UDP** User Datagram Protocol. 3, 4, 7, 9–11

**VM** Virtual Machine. 7