Universiteit van Amsterdam

System and Network Engineering

Research Project 2

Tor: Hidden Service Intelligence Extraction

Author:
João Marques

Supervisors:
Leandro Velasco
Rik van Duijn

August 19, 2018

Abstract

Overlay networks improve and extend the internet functionality without changing the underlying infrastructure. An example of an overlay network is the Tor network, which is focused on improving anonymity and privacy. This research aims to investigate the possibility and efficiency of information extraction from Tor hidden services. The research focuses on the Distributed Hash Table (DHT), where hidden service descriptors are published to. Two servers were used to become part of the DHT and therefore became responsible for descriptors published by hidden services. The process memory and network traffic of these servers were then analysed to find descriptors. Subsequently, the descriptors were extracted and analysed to find the onion address of the hidden service. This was done by implementing a python framework that extracts the full descriptors from memory and processes the independent fields to extract information. With €10,54 per month, the framework was run hourly to extract the onion addresses. A total of 2104 onion addresses were extracted in the process, showing that the majority of the descriptors were using the older v2 protocol instead of the newer v3. Countermeasures such as deprecating the old protocols or the usage of the authentication mechanisms in the old protocol were proposed to solve or help mitigate the issue.

Contents

1 Introduction

With the increase in cybercrime, privacy breaches, and even governmental oppression, there is an increased need for software that is both easy to use and has user security and privacy in mind. The internet and its infrastructure were not designed to provide privacy and anonymity, therefore, overlay networks were created to fulfil these requirements [1]. These networks sit on top of the existing infrastructure and extend it to provide more benefits or missing functionality. This abstracts the underlying network, comes without the need for additional network hardware, and is usually implemented at the application level. One example of such software is the Tor network, which aims to enable its clients to use the Internet with added privacy and anonymity.

The Tor network is a project that started in 2002 [7] and has been under heavy development since then, with new features and improvements being actively released. One such feature is "hidden services", which allows content providers to host their content on the Tor network. These hidden services benefit from the same level of anonymity and privacy as the clients. For hidden services, there is no concept of name resolution (such as DNS) or indexer websites such as Google, Duckduckgo or Bing. This is mainly due to hidden services, as the name implies, being "hidden". Only clients that have received the address through an out-of-band method, are able to find the service by querying the Distributed Hash Table (DHT) for the descriptor, using the onion address. These descriptors contain information on how to contact the hidden service.

Providing anonymity to content providers is an important and welcomed feature. However, by providing anonymity to legitimate content providers it also provides a way to anonymously host illegitimate content such as black markets (for example Silkroad [6]), child pornography, extremist forums, chat platforms, and bot-net C&C (for example Skynet [9]). Consequently, it becomes a double edged sword where the operators of the Tor network want to safeguard the legitimate content while blocking or removing the illegitimate one. This raises yet another problem: any means of finding the illegitimate content can also be used to find the legitimate one, which defeats the purpose of the hidden services since it breaches their anonymity.

Due to the previously mentioned reasons, it is important to know if it is possible to acquire information on the hidden services. Furthermore, if this is possible, it is important to know what the costs of these operations are. In other words, can this be achieved with little funding, or is this something only governmental actors or big companies are able to achieve? This led to the research question proposed for this project:

How efficiently can hidden services information be extracted from the Tor network with limited resources?

To help answer this question the research looked into:

- Can hidden service addresses (onion links) be acquired?
- How feasible is it to act as an hidden service directory (HSDir) server?
- Can descriptor requests and publications be captured in the network?
- Is there a way to capture requests published to an owned HSDir?
- Can information be extracted from descriptors?
- Is is possible to get information from the services themselves once the address is known?

2 Background

An explanation of the general working of the Tor network and hidden services is provided in this section. However, This project focuses specifically on the DHT and its servers, where the descriptors of the service get published.

Overlay Networks are software solutions that are deployed on top of an existent infrastructure that abstracts the underlying network and extends it with new functionality. These networks do this by creating virtual links between the endpoints, abstracting the real network path from the client. This is accomplished by encapsulating the data with information that the overlay network software is capable of recognising, creating a kind of tagging system so that the endpoints know that the packets are destined to them and come from the overlay network. The endpoints can be both physical machines on the underlying network as well as virtual endpoints in software. Furthermore, the virtual links can be encrypted to provide security in transit [5].

Tor Network is an overlay network that provides anonymity and privacy. This is possible due to the Onion Routing Protocol, which makes use of 3 nodes to create a 3 layer encryption of the data being transferred between client and service. The nodes used in this approach are:

- Entry Node (or guard) - The first node on a Tor circuit, relays data between the client and the middle node
- Middle Node - The second node on a Tor circuit relays the client data received from the guard node to the exit node
- Exit Node - The last node, which receives the client data from the middle node, decrypts the last layer of encryption and contacts the service on behalf of the client

With this design, apart of the client, no node (including the service) knows the entirety of the circuit. This design has the aim of defeating tracking attacks where an attacker can see where the data is coming from and going to. By the nodes only knowing the previous and next node in the circuit, an attacker even impersonating a node would not be able to correlate the client and service [18]. The layered approach can be seen below in Figure 1.

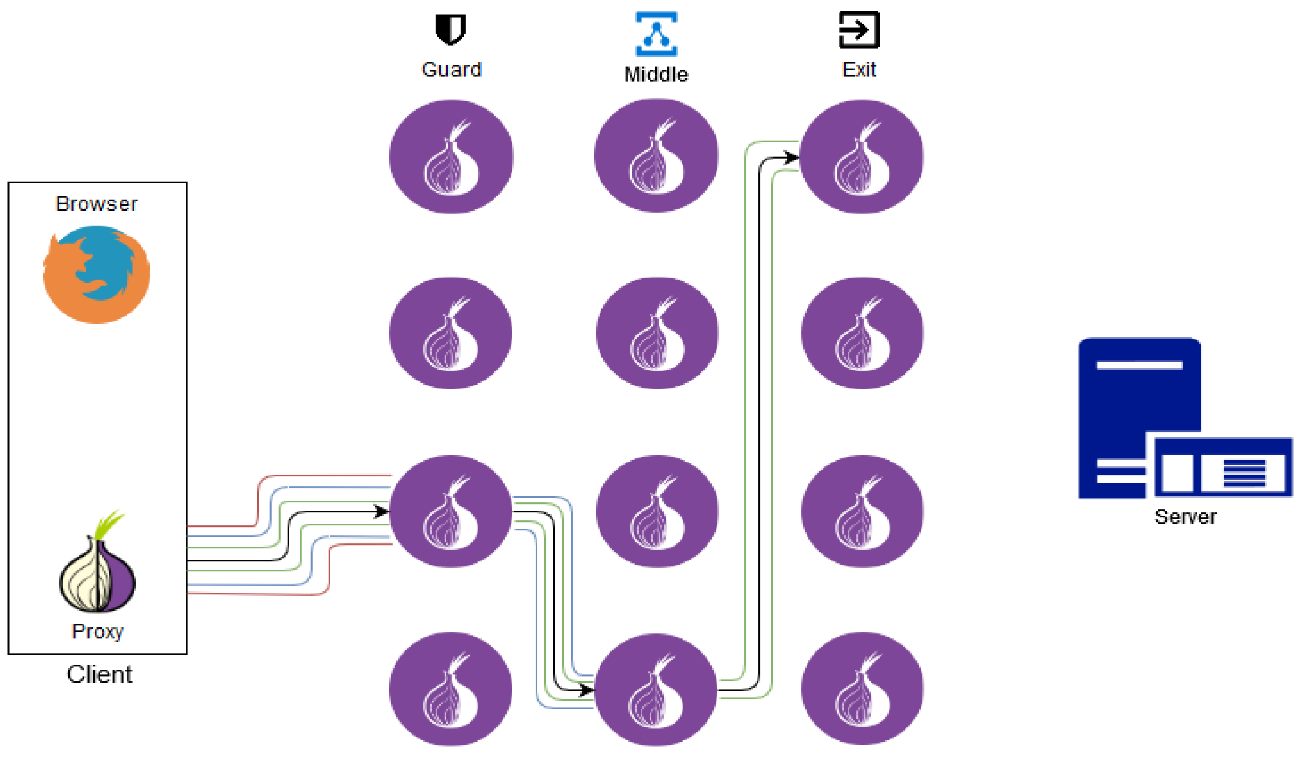


Figure 1: The Onion Routing Protocol layered encryption

Hidden Services is a feature that was released at a later stage of the development of the Tor network and allow the service providers to benefit as well of privacy and anonymity. This is due to the service provider being able to select up to 10 nodes to be the Introduction Point (IP) for its hidden service. Having the IPs keeps the hidden service location from being disclosed. To further anonymise the hidden service each IP is behind a circuit created by the service provider where they are the third node. As a result, the nodes do not know who the service is but know that they need to listen to incoming connections for it and forward through the established circuit, as seen below in Figure 2 [17].

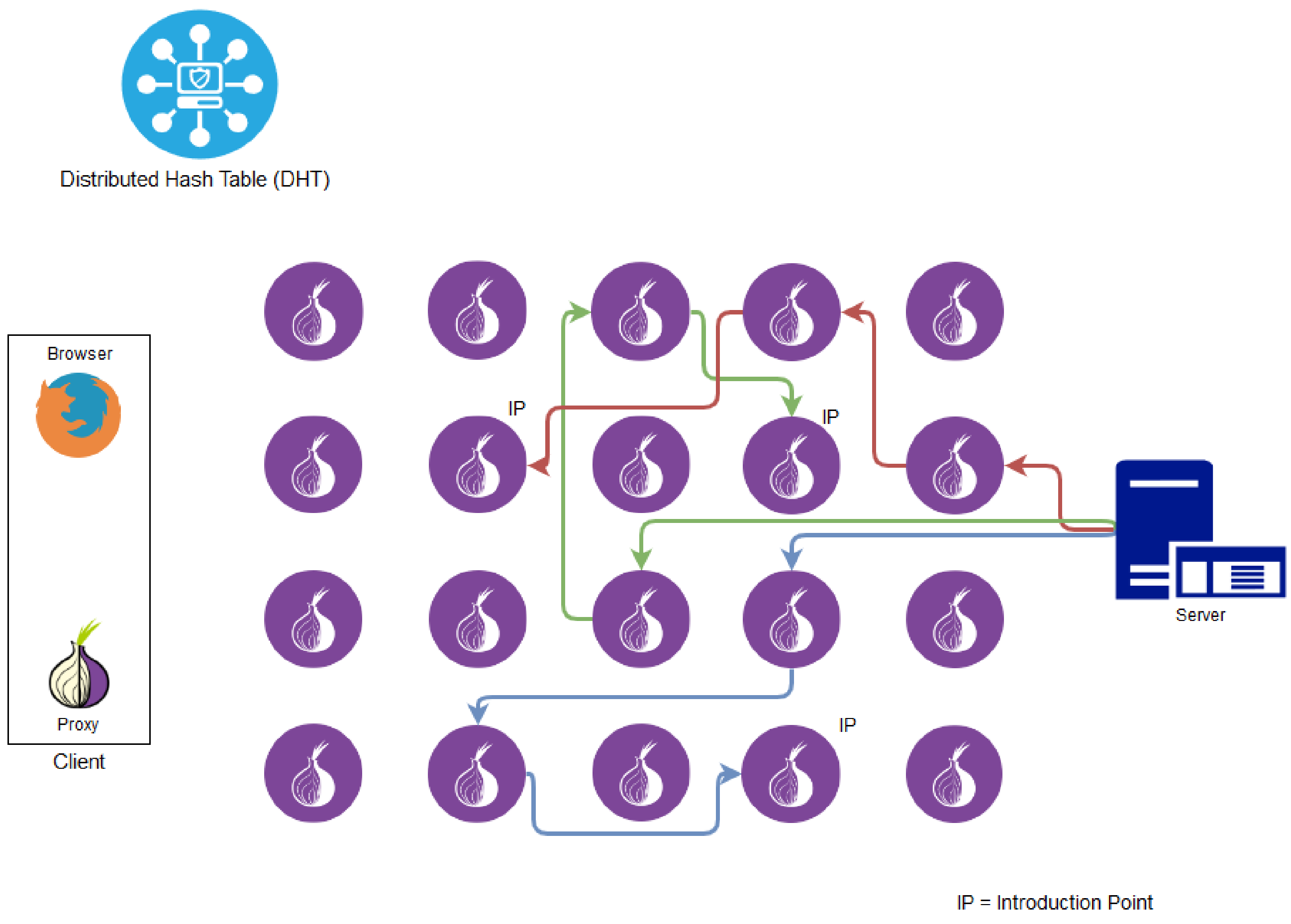


Figure 2: Service choosing its Introduction Points

The server then creates a descriptor which contains information on how to find the introduction points and how to request access to the service. Furthermore, it generates the onion address from the hidden service Public Key. This descriptor is then published to the Distributed Hash Table (DHT) and the onion link is given to clients through an out-of-band method. The DHT is where a group of stable and responsible nodes (nodes with the HSDir and Stable flags) are to receive descriptor publications by the services and descriptor requests by the clients. The HSDir flag indicates that a node is a hidden service directory server from the DHT. Consequently, hidden services can publish service descriptors to it, and client can request the descriptors from it. The stable flag means that the node is suitable for long-lived circuits [16]. Each hidden service gets an ID generated from the encoded onion link with a function of time and the bit 0 at the end. This enables the service to retain the same ID during a period of time. Furthermore, a second ID is generated (replica) where the last bit is flipped to 1. The hidden service server then selects from the DHT the next closest hidden service directory server (HSDir) for each ID (closest fingerprint based on the ID itself). It then publishes the descriptors to each server and to the two servers with the next closest fingerprint. With this process the descriptor is published to a total of six HSDirs, three for the original descriptor and three for the replica to maximise availability. Furthermore, the descriptor ID is subjected to a 24 hours lifetime, after which a new pair is generated and the descriptor is republished with the new ID. The client can then request the descriptor using the onion link of the service. Once in possession of the descriptor, the client chooses a random node on the network to act as a Rendezvous Point (RP). It then creates and keeps a circuit open to the RP. The client then randomly chooses an IP and creates a new circuit to it. Once the circuit is created the client tells the IP that it wants to connect to the specific service on the RP with an arbitrarily chosen cookie. This can be seen below in Figure 3 [17].

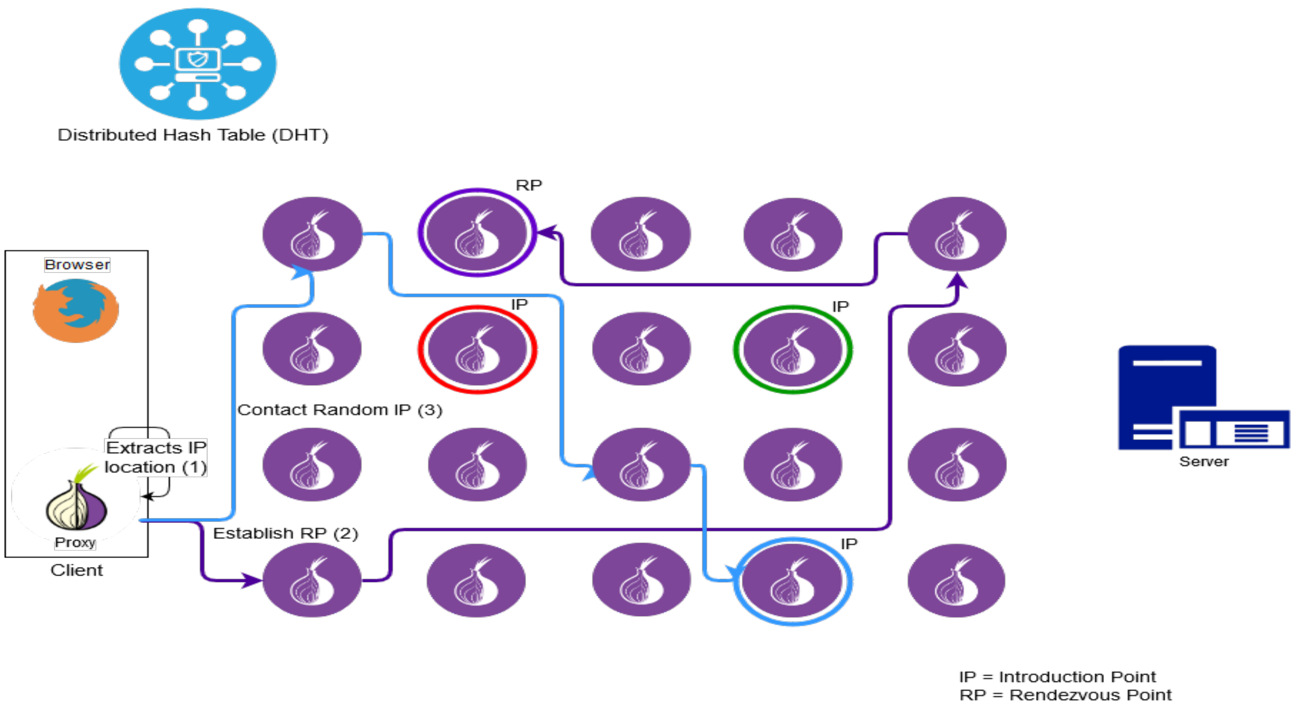


Figure 3: Client retrieving the descriptor and letting the service know what RP to connect to

The IP then relays back the information to the service which in turn opens a 4 node circuit to the RP where the client is already waiting. Once the circuit is established it tells the RP to bridge the connection with the client that is waiting for a connection with the received cookie. The RP then validates the cookie and if it is correct it bridges the connections. At this point, the client and service can communicate back and forth with 6 nodes in between. This can be seen below in Figure 4 [17].

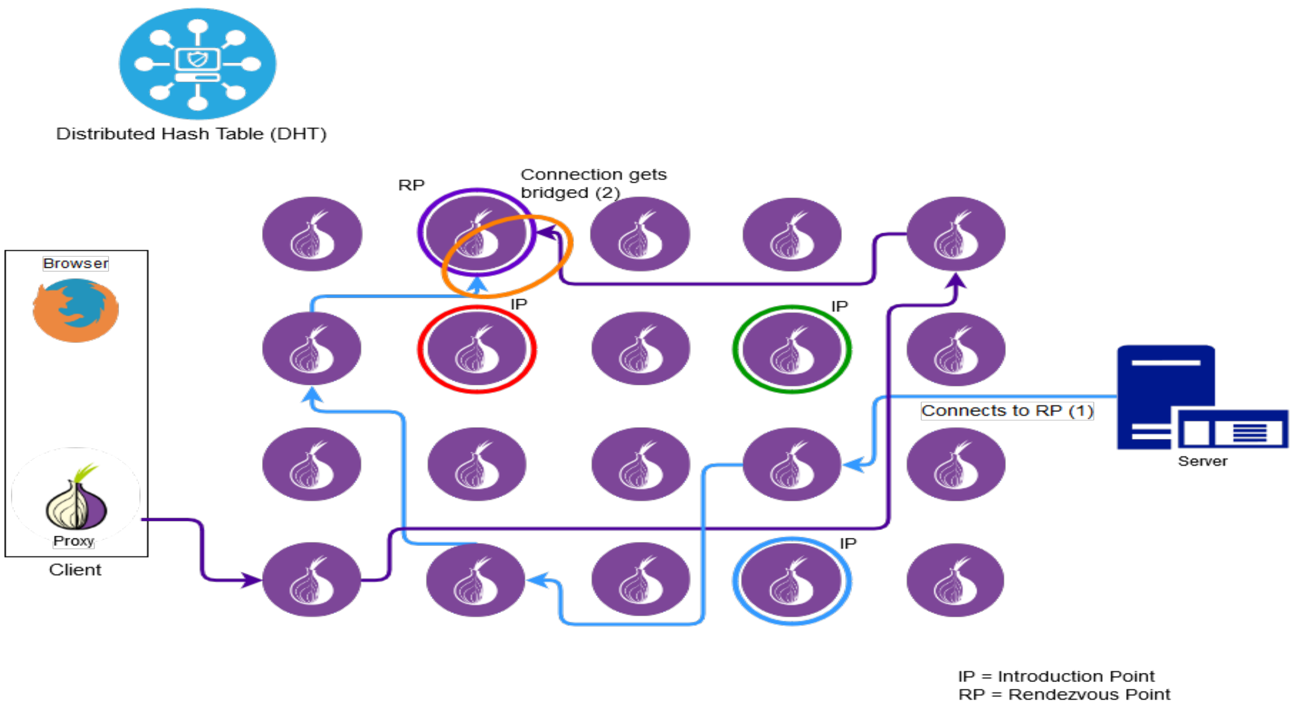


Figure 4: The RP bridges the independent connections between the service and client

3 Literature review

In the past, research has been done into the hidden service functionality of the Tor network. The most notorious paper was written in 2013 by Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann, titled: Trawling for Tor hidden services: Detection, measurement, de-anonymisation [2]. The research was divided into logical stages where first they attempt to retrieve the onion links. Once the links were retrieved, the second stage they attempted to correlate them to requests received by clients. The aim was to statistically analyse and measure the popularity of each service. The third stage was to use the previous stages as a stepping stone to research other techniques of traffic de-anonymisation in the Tor network. The authors also researched an attack where they would calculate the future ID of a hidden service. With this ID, they would then repeatedly generate server fingerprints to obtain a fingerprint in between a hidden service directory server and where the ID would point to in the Distributed Hash Table (DHT). The objective of this operation would be to create servers that would be chosen on the future publication with the calculated ID. Consequently, this would allow the attacker to deny access to the hidden services by not responding to client requests [2]. One thing lacking from this paper is the methodology of the extraction of the onion links or any kind of explanation of how it was done and how one can reproduce the findings.

In 2014 there was a collaboration between Biryukov et al. [2] and a fourth author (Fabrice Thill) to extend the previous research and investigate a new technique called shadowing. This technique makes use of the Tor network limitation of only 2 Tor nodes per IP address and the fact that it keeps track of all extra nodes with the same IP in case the active nodes (nodes with higher bandwidth that are actively relaying connections) go down. With this, it is possible to make the active nodes drop out of the consensus and the shadow nodes (non active nodes) come forward. When the shadow nodes become active, the activity while shadow nodes remain unchanged, allowing the nodes to acquire all of the flags as if they had been active all of the time [3]. Once again no explanation of how the onion links are retrieved, only how the shadow relays can be used to get a large number of HSDir servers.

In 2009 there was a paper by Karsten Loesing titled: Privacy-enhancing technologies for private services [12] which brushes on how hidden services work and what kind of attacks are known to be viable at the time of publication. Furthermore, it describes the at the time current version of the descriptors (v0) and examines a proposal for a new version (v2). It does not mention in any way how to extract onion links but instead, a method is proposed to defeat denial of service attacks on hidden services. This is done by alerting the consensus of directory servers that deny access to a requested descriptor. The consensus then publishes a dummy descriptor and requests it to see if the hidden service directory server does not answer, proving that the HSDir is conducting a denial of service [12].

There were some newer published papers (2017) [8, 19, 14] that use some of the information on the papers mentioned above but nothing that adds to the extraction of onion links or the Intelligence extraction from the services themselves. Furthermore, the research of the older papers is relatively old, when compared to the development rate of the Tor network. This reinforces the need for this research, to verify if it is still possible to achieve the extraction of links, how new developments and implementations change the way this issues can be abused, the impact and feasibility, and finally to review what countermeasures can be implemented.

4 Methodology

In this section, the method used for this research will be explained. How the environment was set up, what experiments were conducted, their results and the reasoning behind them. Furthermore, the python framework developed for this research is also introduced and explained how it wraps the different tools used to extract the onion links and store them. Finally, an assessment of the costs and achievements is done at the end to answer the research question, **How efficiently can hidden services information be extracted from the Tor network with limited resources?**.

4.1 Experiment Environment

For this research two servers where used with different specifications to assess the performance impact on the onion link extraction and membership to the DHT. The hardware used to conduct this project can be found in Table 1 and the software in Table 2 below:

Hardware	**OS**	**RAM**	**Network**
Namecheap VPS (Pulsar)	Ubuntu 18.04 (64bit)	2 GB	1000 GB/Month
OS3 (University Server)	Ubuntu 17.10 (64bit)	8 GB	Unlimited Traffic

Table 1: Hardware used for the research

Software	**Version**
Tor (Proxy)	0.3.3.7
Nyx (Tor Monitoring)	2.0.4
Python	3.6.3
GDB	8.0.1

Table 2: Software used for the research

The two servers were first secured by configuring the firewall and then the Tor proxy software was installed. Moreover, the ports 443 and 9030 were opened in the firewall to allow for the Tor software to be able to operate. Tor on each server was configured using one of the the two configurations found in Appendix A. These configurations are very basic containing the bare minimum to run a non-exit Tor relay. The only noteworthy differences are: the Namecheap configuration file contains heavier bandwidth constraints and accounting is enabled to disable the relay once it relays 900 gigabytes of data. Furthermore, the configuration files contain lines to allow for monitoring tools to connect to it on a specific port and an authentication based on a predefined cookie.

Once both server were acting as Tor relays, an instance of Nyx was started in each server to monitor them using the control port (9051) which was accessible through the loopback interface. Nyx enabled the monitoring not only of the proxy relaying bandwidth usage but also the acquisition of flags. This was helpful to document how much time each flag took to be acquired and the effects of the relay crashing or being restarted. Furthermore, it enabled the sending of commands directly to the command port. This allowed for the requesting and logging of descriptors for a first analysis of its contents. Unfortunately, the logging is only due to the proxy acting as a client and directly requesting a descriptor from the responsible HSDir, as a result it needs to receive an onion address to be able to make the request.

The next step was to wait for the required flags, HSDir and Stable, to be delegated to the servers. The HSDir flag took 4 days to obtain after the servers obtained the Stable flag. On average, this took 5 days of continuous running. Once the Stable flag is obtained for the first time the HSDir flag also gets delegated right away. Both servers were able to get the HSDir flag despite the difference in specifications and constraints imposed on the relay bandwidth.

As mentioned in the background section, the server with the HSDir flag now is able to receive descriptor publications from hidden services. The publications identifier changes every 24 hours, meaning that the responsible HSDirs can change once the descriptor refreshes the identifier. Moreover, the refresh time is not synchronised between hidden services, and for this reason there can be HSDir changes through out the day. As a result, a single server could receive many publications through out the day.

4.2 Approaches

For this project, 5 possible approaches were considered: modifying the Tor software, internet scraping, address bruteforcing, network sniffing, and memory extraction. The approaches that were chosen for this research were network sniffing and memory extraction, the motivation for this choice can be seen bellow.

Modification of the Tor software would be a viable option due to it being open source code. This approach would allow for a clean method of capturing descriptors and client requests into an easy and accessible format for processing. The disadvantages of this approach are the time constraints of the project coupled with the lack of knowledge of the "C" programming language. Furthermore, if the software would be successfully modified, it would require high maintenance to keep on track with changes to the original code and changes to the underlying infrastructure. As a result, it was decided to not pursue this approach and focus on other approaches.

Internet scraping, besides being time-consuming, requires the links to have been shared on the internet in clear text. Although this is the case for a large portion of the created hidden services, it does not allow for the capture of the ones that are not shared or are for private use.

Bruteforcing could be a valid method to find hidden services due to the small size of v2 addresses (16 characters links). However, it is unfeasible due to being time-consuming and requiring the descriptor to be fetched to confirm it is valid which would further slow down the process.

Network sniffing is a method to capture the descriptors when they get published by the hidden service or requested by the client. When the client requests a descriptor it uses an ID derived from the onion address using a hashing mechanism. Although, it is not possible to retrieve the address from the ID due to the hashing mechanism, it should be possible to see the response of the proxy which would contain the requested descriptor. This led to the investigation of the possibility of capturing these interactions between hidden services or clients, and the HSDir.

The memory extraction is a reliable method due to the HSDir server needing to hold the descriptors so clients can request them. Since it is a security risk to write them to the filesystem, the application keeps it in the process memory. With privileges, this memory can be easily accessed and dumped for analysis.

4.2.1 Network sniffing

The first method attempted was network sniffing for incoming HSDir descriptor requests and publications. For this tcpdump was used to dump all traffic coming to the Internet-facing interface. This traffic was then piped into grep with different search patterns for terms taken from the Tor specification as well as other more generalised terms for extraction control. The patterns include: "GET /tor/rendezvous/", "GET /tor/rendezvous2", "GET /tor/rendezvous2/publish", and "GET /tor/hs". To test the receiving of other resources as a control capture it was also attempted with just "GET". These patterns were chosen in accordance with the protocol specification of how requests and publications are performed. The control pattern was chosen due to the server making and receiving general resource requests with that pattern. The experiment was run for around 20 hours.

After 20 hours of network sniffing, the experiment only captured "GET" requests for general Tor resources. None of the expected patterns captured descriptors being published or requested by clients. The resources captured seem to be resources for circuit creation. This experiment was being conducted at the same time that descriptors were being extracted from memory. As a result of the extraction of memory it was shown that several new publications where being received through out the 20 hours of the network sniffing experiment.

The problem with the network sniffing method can be attributed to one or more reasons. First, it could be due to the client caching the descriptor [10]. Although this would seem strange, it could somewhat explain the lack of requests for descriptors. Furthermore, hidden services descriptors only stay on a server 24 hours before a new id is calculated and the descriptor republished to a new set of HSDirs. They get published the first time and only need republishing within the 24 hours if any introduction points are added, removed, or changed [3]. Moreover, the period where descriptors get republished is not synchronised. Consequently, new descriptors can be received at any time of the day. Furthermore, this experiment was conducted at the same time as an hourly descriptor extraction from memory. In the results of the memory extraction for the period of the network sniffing, there were at least 1 new descriptors published per hour.

Another reason for not being able to capture the network traffic could be attributed to the fact that clients create circuits to the hidden service [16]. This could mean that since the encryption keys are negotiated with the application, sniffing the traffic which is done at the network layer would show only encrypted blobs, and not trigger the search patterns. Furthermore, the resources captured with the control pattern were resources required for normal operation of the proxy which could indicate that they are not fetched through circuits, but instead directly. Resources such as the consensus file which contains the nodes to create circuits. If this is the case, then the only possible way to capture the requests and publications is if the proxy would be modified to store that data, in this way bypassing the encryption present on the network layer.

Lastly, the least plausible reason, the specification describes the descriptor requests and publications as HTTP get and post requests [16]. It could be that changes were made and the corresponding specification file has not been updated to reflect it. This could explain why the terms used did not yield any results.

Even if the capturing had been successful, the captured information would not be directly possible to extract onion links. In the case of v2 and v3 descriptors, this could be due to the client using the ID which can only be calculated from the onion address but not back since it has an hashing mechanism step. However, it is possible to use the captured ID to request the descriptor from another responsible server or capture the response to the request which would contain the descriptor itself. Another use is to correlate requests to previously captured descriptors. With captured descriptors, one can extract the onion address and then calculate the ID and correlate with capture ID's to analyse the popularity of the hidden service. This would be useful for example to assess the size of bot-nets C&C which have become quite popular to host in the Tor network [4].

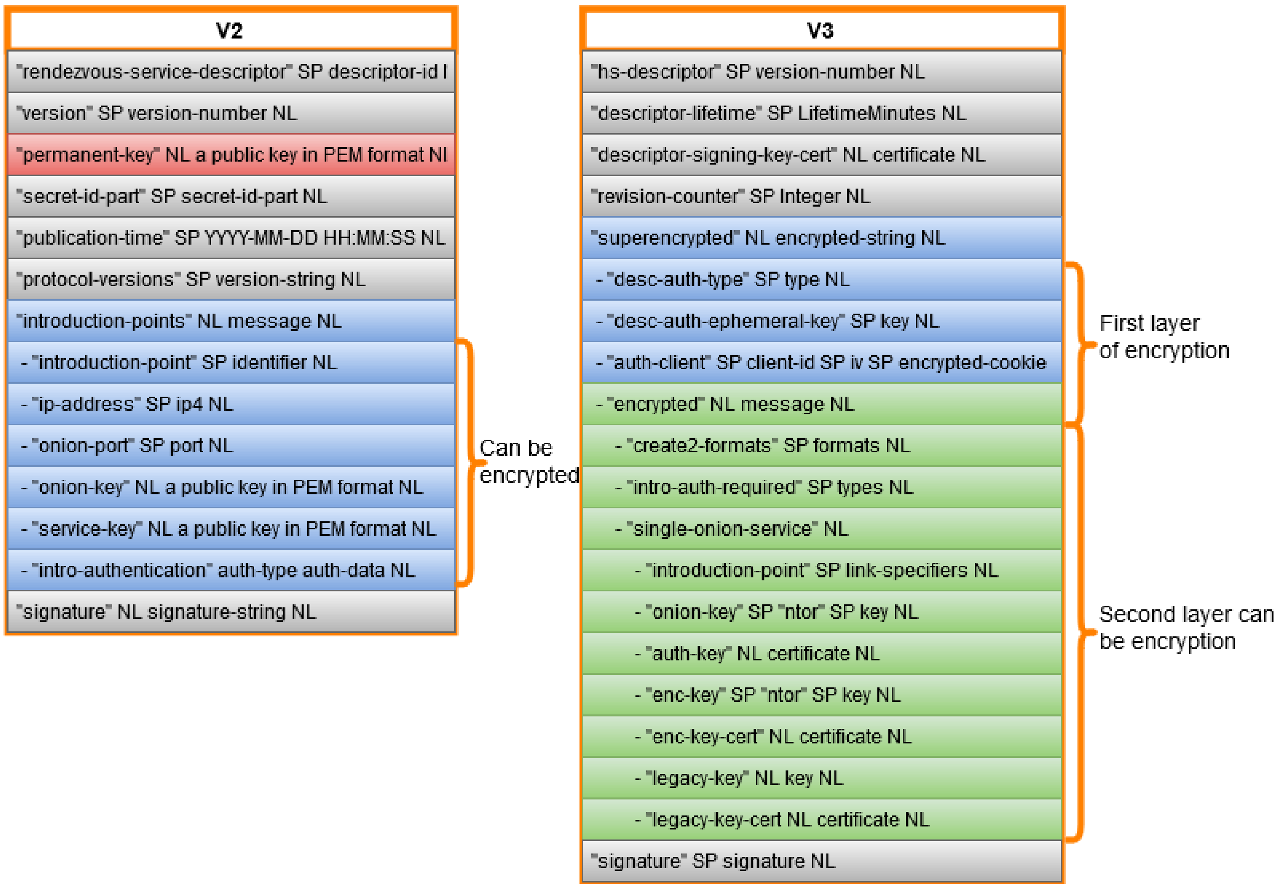
4.2.2 Memory Extraction

For the memory extraction method the memory mappings of the Tor process needed to be found. This was done by listing the contents of the "maps" file in the "/proc/<pid>" tree. The contents needed to be filtered for mappings with read/write permissions since the data being extracted needed to be written and read on the fly. With those mappings it then was possible to extract the start and end memory addresses and pass them to GDB (Debugger) to dump everything in between. The following Listing 1 shows the command used to accomplish this in an automated way:

```
grep rw-p /proc/"$(pgrep '^tor')"/maps |
grep -v "lib" |
sed -n 's/^\([0-9a-f]*\)-\([0-9a-f]*\) .*$/\1 \2/p' |
while read start stop; do
    gdb --batch --pid "$(pgrep '^tor')" -ex "dump memory
     ↪  $start-$stop.dump 0x$start 0x$stop" &> /dev/null;
done;
```

Listing 1: Command to automatically dump memory from each mapping of a process

The dumped memory sections where then analysed with the "Strings" utility. This utility extracts every string with more than 4 characters, which conforms with the descriptor specification. The strings utility would be run in all of the memory section files and all of the outputted strings would then be gathered in a single file for easier analysis. The descriptors could then be found within the strings file using the following structure shown in Figure 5 below:



V2

"rendezvous-service-descriptor" SP descriptor-id I
"version" SP version-number NL
"permanent-key" NL a public key in PEM format NI
"secret-id-part" SP secret-id-part NL
"publication-time" SP YYYY-MM-DD HH:MM:SS NL
"protocol-versions" SP version-string NL
"introduction-points" NL message NL
- "introduction-point" SP identifier NL
- "ip-address" SP ip4 NL
- "onion-port" SP port NL
- "onion-key" NL a public key in PEM format NL
- "service-key" NL a public key in PEM format NL
- "intro-authentication" auth-type auth-data NL
"signature" NL signature-string NL

V3

"hs-descriptor" SP version-number NL
"descriptor-lifetime" SP LifetimeMinutes NL
"descriptor-signing-key-cert" NL certificate NL
"revision-counter" SP Integer NL
"superencrypted" NL encrypted-string NL
- "desc-auth-type" SP type NL
- "desc-auth-ephemeral-key" SP key NL
- "auth-client" SP client-id SP iv SP encrypted-cookie
- "encrypted" NL message NL
- "create2-formats" SP formats NL
- "intro-auth-required" SP types NL
- "single-onion-service" NL
- "introduction-point" SP link-specifiers NL
- "onion-key" SP "ntor" SP key NL
- "auth-key" NL certificate NL
- "enc-key" SP "ntor" SP key NL
- "enc-key-cert" NL certificate NL
- "legacy-key" NL key NL
- "legacy-key-cert NL certificate NL
"signature" SP signature NL

Figure 5: Descriptor structure for both versions of the descriptor in the current specification [16]

On the V2 descriptor, the "permanent-key" field can be found. This field contains the public key of the hidden service which is used to establish the communication's ephemeral encryption key. Due to the onion address generation being based on the public key, any party that has access to the public key can also generate the onion address. The onion address is generated in the following way; the Public key is first base64 decoded. It is then passed through the SHA1 hashing function and only the first 80bits (half of the output) is retained. Finally, these 80 bits are encoded in base32 and the output turned to lower-case. Using the structure and the address generation process, it was possible to pass each v2 descriptor's "permanent-key" (public key) through the following command in Listing 2 below, to extract the onion link:

```
echo "<perm-key value>" | base64 -d | sha1sum - | cut -c 1-20 |
↪  xxd -r -p | base32 - | tr 'A-Z' 'a-z'
```

Listing 2: Command to automatically extract the onion links from the public key of the descriptor

The memory extraction approach successfully and consistently extracted descriptors. Descriptors can be reliably extracted by following the structure presented in the specification [16]. Onion addresses could be extracted from v2 descriptors without any issues. It was not possible to extract the onion addresses from v3 descriptors.

The v3 protocol aims to solve the problem of the HSDirs being able to extract the onion address from the descriptors published to them. The v2 protocol, opposed to v3, contains the public key of the service in the descriptor, which enables anyone with access to the descriptor to decode the onion address. This can be seen in Figure 5 where the v2 has the permanent-key (red field). On the other hand, the v3 adds more layers of encryption, leaving only the metadata necessary for identification of the descriptor unencrypted. Furthermore, v3 does not contain the public key, not even in the encrypted section [16]. During the experiment, it was possible to see more v2 descriptors than v3 in memory at any single time. This could show that there is still a higher adoption of the v2 protocol if it could be constantly seen throughout a longer period of time. This will be assessed in the next subsection where the framework is explained.

4.3 Python Framework

To automate the whole process and add further processing of the descriptors, a python framework was created to wrap all of the separate tasks. The source code for the entirety of the project can be found on the project's GitHub page[1]. Furthermore, to have both servers writing to the same database the data processed from the descriptors, the directory containing the database file was mounted from one server to the other using ssh file system (sshfs). The tools were then set to run as hourly cronjobs: The OS3 server at 15 minutes past the hour and the Namecheap server at 45 minutes past the hour. An overview of the framework can be found below in Figure 6.

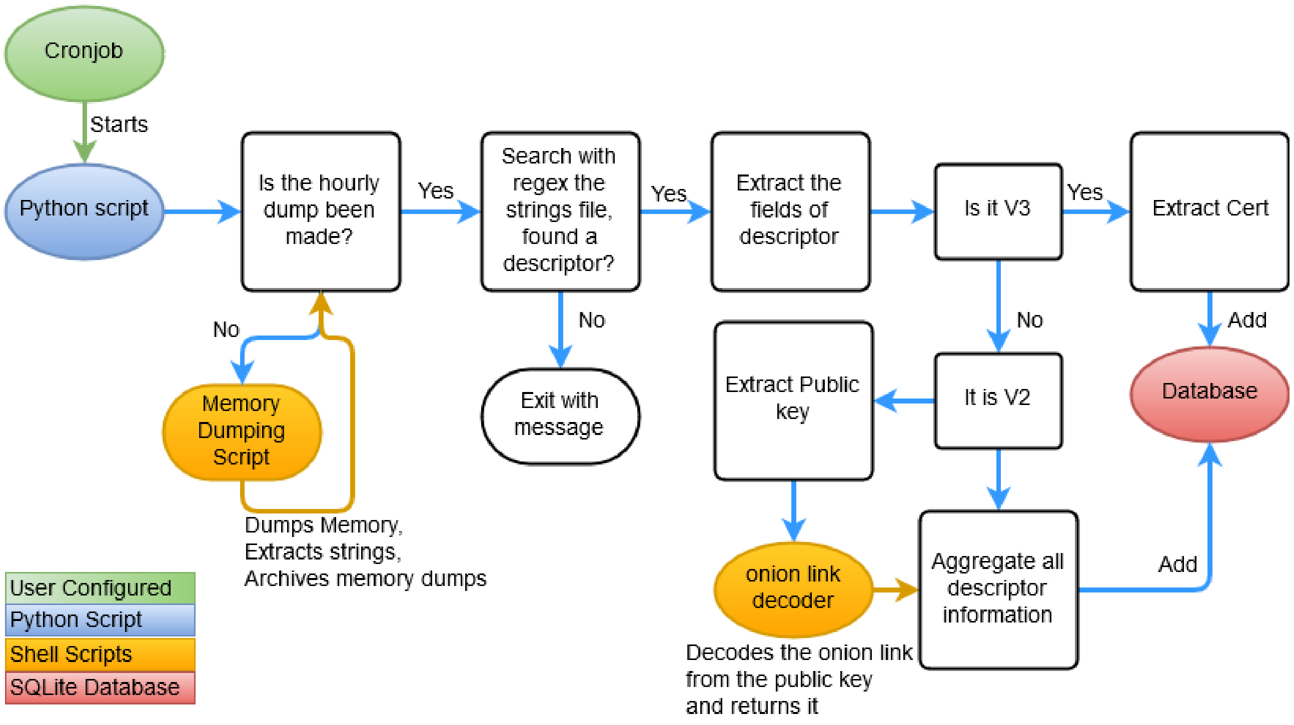

Cronjob
Starts
Python script
Is the hourly dump been made?
Yes
No
Memory Dumping Script
Dumps Memory, Extracts strings, Archives memory dumps
Search with regex the strings file, found a descriptor?
Yes
No
Exit with message
Extract the fields of descriptor
Is it V3
Yes
Extract Cert
Add
Database
No
It is V2
Extract Public key
onion link decoder
Decodes the onion link from the public key and returns it
Aggregate all descriptor information
Add
User Configured
Python Script
Shell Scripts
SQLite Database

Figure 6: Process diagram of the python framework that extracts and processes descriptors directly from an HSDir process memory

Due to the sensitive nature of the data captured, all of the acquired data did not leave the servers where they were captured. Furthermore, once the experimentation was done and documented, the servers databases and any other sensitive data were removed in a secure manner to limit the possibilities of recovery as much as possible.

[1]https://github.com/JCMarques15/tor-hs-fetcher

The automated tool was set to run as an hourly cronjob and in 5 days the tool captured 171 v3 descriptors and 2104 v2 (12.3 times more than v3). This shows that there has not been a big adoption of the v3 protocol.

The resulting captures where only from the OS3 server, due to a race condition problem with the Namecheap server. This is due to the Namecheap server being located in the United States. The servers geolocation and lower processing power became an obstacle when it tried to extract the memory, process it, and write to the same database as the OS3. The server would start to write to the database around the same time as the next run of the OS3 server. The OS3 server with superior processing power and writing locally advantage would lock the database and force the Namecheap server to time-out. Once timed-out the framework would exit and clear all of the data to not create partial commits. This problem could have been easily mitigated with optimisation to the starting time or with the separation of the database and with a mechanism to gather all of the data into a single database post run. Either way, it did not affect the purpose of the research and shows how much it can capture with few resources.

The cost of this experiment was 1 server for €10,54, capturing for 5 days. When comparing these findings to the Tor metric website that shows 99737 unique hidden services (as of 2018-07-22) which is 47.4 times more than what was captured. It can be extrapolated that with more time or servers, including 2 nodes per IP which were not tested during this experiment, could bring the captured value close to the reported by the Tor metrics. The Tor metrics extrapolates this values from obfuscated statistics reported by a small subset of nodes. This subset are nodes that have configured the proxy to send these statistics, which come turned off by default [15]. Due to a hidden service onion address not changing, long lived hidden services will keep the same onion address no matter how many times the descriptor ID is refreshed. As a result, an HSDir could end up being chosen as one of the six HSDirs each day. Once the descriptor and subsequently the onion address gets extracted, the link would stay valid as long as the hidden service is not shut down.

5 Countermeasures

Despite the research revolving around the hunting of illegal content services and their monitoring, it is against the core principles of the Tor network which are to provide privacy and anonymity to its users. To do this there are some measures that can be applied by the users and some by the Tor network itself. The best course of action would be to deprecate the v2 descriptor in the same way that the v0 was deprecated before in favour of the v2 and adopt the more secure v3 protocol. The v3 protocol despite the increased character link is generated from better cryptographic algorithms as well as make use of better hashing and negotiation algorithms for its normal operation (SHA1/DH/RSA1024 to SHA3/ed25519/curve25519)[13]. Despite the benefits, phasing out the v2 protocol is not an easy accomplishment due to the number of services that still rely on it.

Another solution would be new service providers generating v3 services as opposed to v2. The problem with the usage of v3 is users having to remember or keep a record of 56 character links instead of the easier 16 characters. Moreover, some services rely on the shorter address of v2 (16 characters opposed to the v3 56 characters) to generate services with custom names (brute-forcing the generation until it yields a result closed to the desired) using tools such as eschalot [21] and scallion [20], which are easier for a user to remember, for example the facebookcorewwwi.onion.

When all of these are not possible, it is recommended that hidden services make use of authentication to authenticate its users with a password or cookie to reduce the attack surface of an attacker that has acquired the onion address. This would improve the security by encrypting the introduction points which can only be accessed by authorised users. Without introduction points even if a server has access to the descriptors and subsequently the onion link, it would not be able to use it as a stepping stone to launch attacks or gather intelligence on the hidden service due to the authentication requirements.

6 Future Work

This research showed that it is possible to extract onion links using low cost hardware with relative ease. However, this research can be used as a stepping stone for further research into the security of the services.

The v3 protocol is relatively recent so it would be interesting to look further into it and analyse if the cryptographic operations are well implemented and if it is truly impossible to recover the onion link. Furthermore, investigating the cryptographic operations could uncover other security flaws.

Another interesting area of research would be to investigate if it is possible to extract information from the service provider's servers, as well as analysing how many hidden services actually use the client authentication methods. A starting point would be to find a way to differentiate encrypted introduction points from bad encoding (for example from old partially overwritten descriptors still in memory).

The v2 protocol authentication method could also be analysed further for means of exploitation, such as bruteforcing. The v2 encryption algorithms are relatively weak and could possibly be broken. This could emphasise the need for the adoption of the v3 protocol even further.

Finally, it would be interesting to assess how many Hidden Services are short-lived on-demand creations from for example secure file transfers such as OnionShare [11]. These hidden services would need to be monitored for "aliveness" and possibly even scanned to figure the kind of service. This could give insight into the size of the subset of long lived onion addresses.

7 Conclusion

Nowadays, privacy and anonymity online is greatly increased by the use of overlay networks, such as the Tor network. Some of the Tor network features also provide anonymity to service providers, which can be taken advantage of for illegitimate usage. This raises the need to secure legitimate content and monitor or block illegitimate content. However, this can be complicated by privacy and usage policies of the Tor network. One way to find these hidden services, to assess the legitimacy of the content, is through their onion address which are not always shared publicly. For this reason, this research was conducted to examine the possibilities of acquiring onion addresses as well as analysing the cost. The research shows that it is possible to gather thousands of onion addresses through memory extraction of servers belonging to the Distributed Hash Table in a small amount of time. Furthermore, it shows it can be done with cheap resources and a small amount of time. These findings can easily be used as a stepping stone for further attacks and intelligence extraction from the Tor network users. For this reason, it is imperative that the newer v3 protocol becomes more widely used, or that the security features of the v2 protocol are implemented. This project shows the fundamental security and privacy problems of the v2 protocol and shows the necessity for improvement in this area to protect the anonymity of the users.

References

[1] Christer Andersson. “Design and evaluation of anonymity solutions for mobile networks”. PhD thesis. Fakulteten för ekonomi, kommunikation och IT, 2007.

[2] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. “Trawling for tor hidden services: Detection, measurement, deanonymization”. In: *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE. 2013, pp. 80–94.

[3] Alex Biryukov et al. “Content and popularity analysis of Tor hidden services”. In: *Distributed Computing Systems Workshops (ICDCSW), 2014 IEEE 34th International Conference on*. IEEE. 2014, pp. 188–193.

[4] Matteo Casenove and Armando Miraglia. “Botnet over Tor: The illusion of hiding”. In: *Cyber Conflict (CyCon 2014), 2014 6th International Conference On*. IEEE. 2014, pp. 273–282.

[5] SDX Central. *What is Overlay Networking (SDN Overlay)?* URL: `https://www.sdxcentral.com/sdn/definitions/what-is-overlay-networking/`.

[6] 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*. ACM. 2013, pp. 213–224.

[7] Roger Dingledine. *pre-alpha: run an onion proxy now!* 2002. URL: `https://lists.torproject.org/pipermail/tor-dev/2002-September/002374.html`.

[8] Shalini Ghosh et al. “Automated Categorization of Onion Sites for Analyzing the Darkweb Ecosystem”. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2017, pp. 1793–1802.

[9] C Skynet GUARNIERI. *a Tor-powered botnet straight from Reddit*. 2012.

[10] Nicholas Hopper. “Challenges in protecting tor hidden services from botnet abuse”. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2014, pp. 316–325.

[11] Micah Lee. *Securely and anonymously share a file of any size*. 2018. URL: `https://github.com/micahflee/onionshare`.

[12] Karsten Loesing. *Privacy-enhancing technologies for private services*. Vol. 2. University of Bamberg Press, 2009.

[13] nickm. *Tor 0.3.2.1-alpha is released, with support for next-gen onion services and KIST scheduler*. 2017. URL: `https://blog.torproject.org/tor-0321-alpha-released-support-next-gen-onion-services-and-kist-schedulerd`.

[14] Andriy Panchenko et al. “Analysis of fingerprinting techniques for tor hidden services”. In: *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*. ACM. 2017, pp. 165–175.

[15] The Tor Project. *Tor metrics of v2 hidden services*. 2018. URL: `https://metrics.torproject.org/hidserv-dir-onions-seen.html`.

[16] The Tor Project. *Tor Specifications*. 2018. URL: `https://gitweb.torproject.org/torspec.git/tree/`.

[17] Tor Project. *Tor: Onion Service Protocol*. URL: `https://www.torproject.org/docs/onion-services.html.en`.

[18] Tor Project. *Tor Project: Overview*. URL: `https://www.torproject.org/about/overview.html.en`.

[19] Iskander Sanchez-Rola, Davide Balzarotti, and Igor Santos. “The onions have eyes: A comprehensive structure and privacy analysis of tor hidden services”. In: *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2017, pp. 1251–1260.

[20] Eric Swanson. *GPU-based Onion Hash generator*. 2017. URL: `https://github.com/lachesis/scallion`.

[21] *TOR hidden service name generator*. 2017. URL: `https://github.com/ReclaimYourPrivacy/eschalot`.

Appendices

A Tor Configuration Files

```
OS3: /etc/tor/torrc
  Nickname <proxy-name>
  ContactInfo <operator-email>

  ORPort 443
  DirPort 9030
  ExitRelay 0
  SocksPort 0
  ControlSocket 0

  BandwidthRate 10 MB
  BandwidthBurst 12.5 MB
  RelayBandwidthRate 10 MB
  RelayBandwidthBurst 12.5 MB

  DisableDebuggerAttachment 0
  ControlPort 9051
  CookieAuthentication 1
```

Listing 1: OS3 Tor Configuration

```
Namecheap: /etc/tor/torrc
  Nickname <proxy-name>
  ContactInfo <operator-email>

  ORPort 443
  DirPort 9030
  ExitRelay 0
  SocksPort 0
  ControlSocket 0

  AccountingStart month 1 00:00
  AccountingMax 900 GB
  RelayBandwidthRate 300 KB
  RelayBandwidthBurst 340 KB

  DisableDebuggerAttachment 0
  ControlPort 9051
  CookieAuthentication 1
```

Listing 2: Namecheap Tor Configuration