
Universiteit van Amsterdam

Systems and Network Engineering


Research Project 2

# Tor: Revealing the Hidden Shallots


*Author:*
João Marques

*Supervisors:*
Leandro Velasco
Rik van Duijn


July 30, 2018


**Abstract**

With the current need for privacy and anonymity, overlay networks such as the tor network have been created. With the development of the tor network, not only users but service providers can benefit from these values. But how safe are the anonymity mechanisms? This research aims to investigate the **possibility and efficiency of information extraction from hidden services**. To accomplish this, the research focused on the Distributed Hash Table (DHT) where hidden service descriptors are published to. Two servers were used to become part of the DHT and in turn responsible for descriptor 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 less then 24€/month for 5 days, the framework was run hourly to extract the onion addresses. A large number of 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. Further research could also be conducted to assess the benefits of the newer protocols and methods devised to phase out the older protocol without large disruptions.

# 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. Due to the Internet and its infrastructure not having been designed with these values, overlay networks were created. These networks sit on top of the existing infrastructure and extend it to provide more benefits or missing functionality. This abstracts the underlying network and comes without the need for additional network hardware, usually being 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 [6] and has been under heavy development since then, with new features and fixes for current ones being actively released. One such feature that was released is "Hidden Services" which allow content providers to host their content on the Tor Network. These Hidden Services benefit from the same anonymity and privacy as the clients. For Hidden Services, there is no concept of name resolution (DNS) or indexer websites such as Google, Duckduckgo, Bing, etc... This is mainly due to Hidden Services, as the name implies, being "hidden". Only clients that have received with an off-band method the address, are able to find the service by querying the DHT for the descriptor of the onion address. These descriptors contain information of how to contact the Hidden Service.

Providing anonymity to content providers is an important and welcomed feature. But by providing anonymity to legitimate content providers it also provides a way to anonymously host illegitimate content such as black markets (exp. Silkroad [5]), child pornography, extremist forums/chat platforms, bot-net C&C (exp. Skynet [7]), etc... So it becomes a double edge sword where the operators of the tor network want to safeguard the legitimate content while blocking/removing the illegitimate one. Which raises 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.

So for the reasons above it was deemed important to know if it is even possible to acquire information on the hidden services. If so how much would the method cost, if it is accessible to low funding or if its something only governmental actors are able to achieve. So the research question proposed for this research was the following:

**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 be acquired?
- How feasible is to act as an HSDir (Hidden Service Directory Server)?
- Can descriptors requests/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?

Although, this research focuses specifically on the Distributed Hash Table (DHT) and its servers, where the descriptors of the service get published. An explanation of the general working of the tor network and Hidden Services are provided in the background section.

# 2 Background

**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 for the client. This is accomplished by encapsulating the data with information that the overlay network software is capable of recognizing, 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 [4].

**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, no node (including client and 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 [15]. The layered approach can be seen below in figure 1.

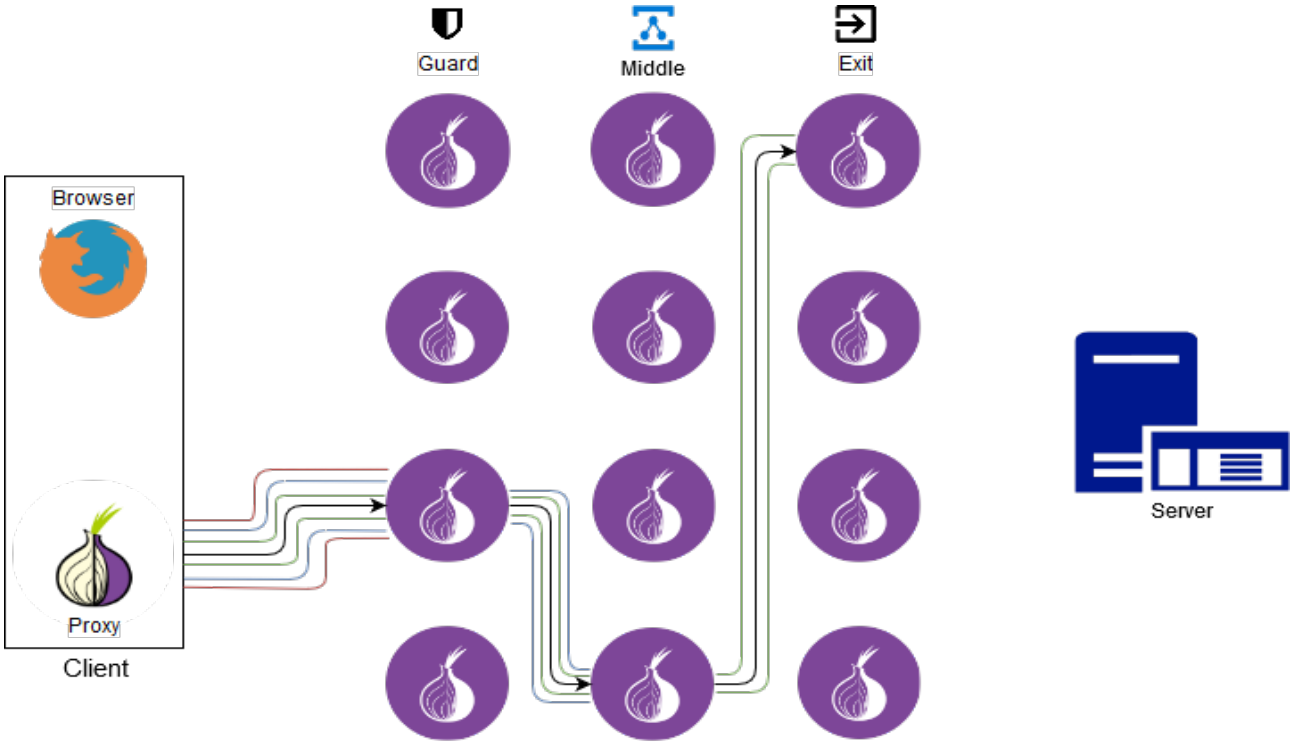


Figure 1: The Onion Routing Protocol layered encryption

**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. So 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 [14].

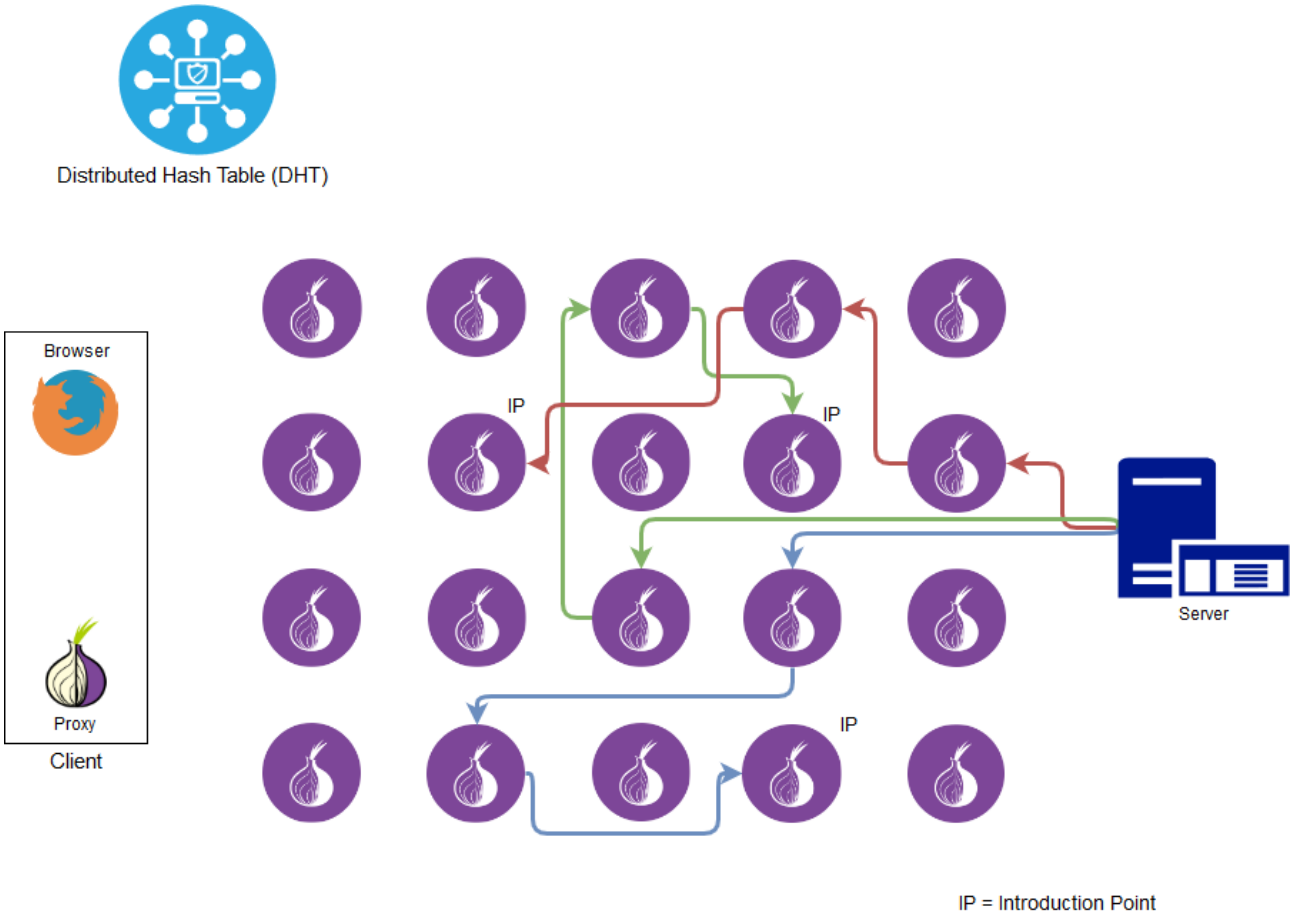


Figure 2: Service choosing its Introduction Points

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. These descriptors are then published to the Distributed Hash Table (DHT) and the onion link given to clients through an off-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 requests by the clients. Once the descriptor is published the clients can request the descriptor provided that they know 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), creates and keeps a circuit open to it. It then randomly chooses an IP and creates a 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 [14].

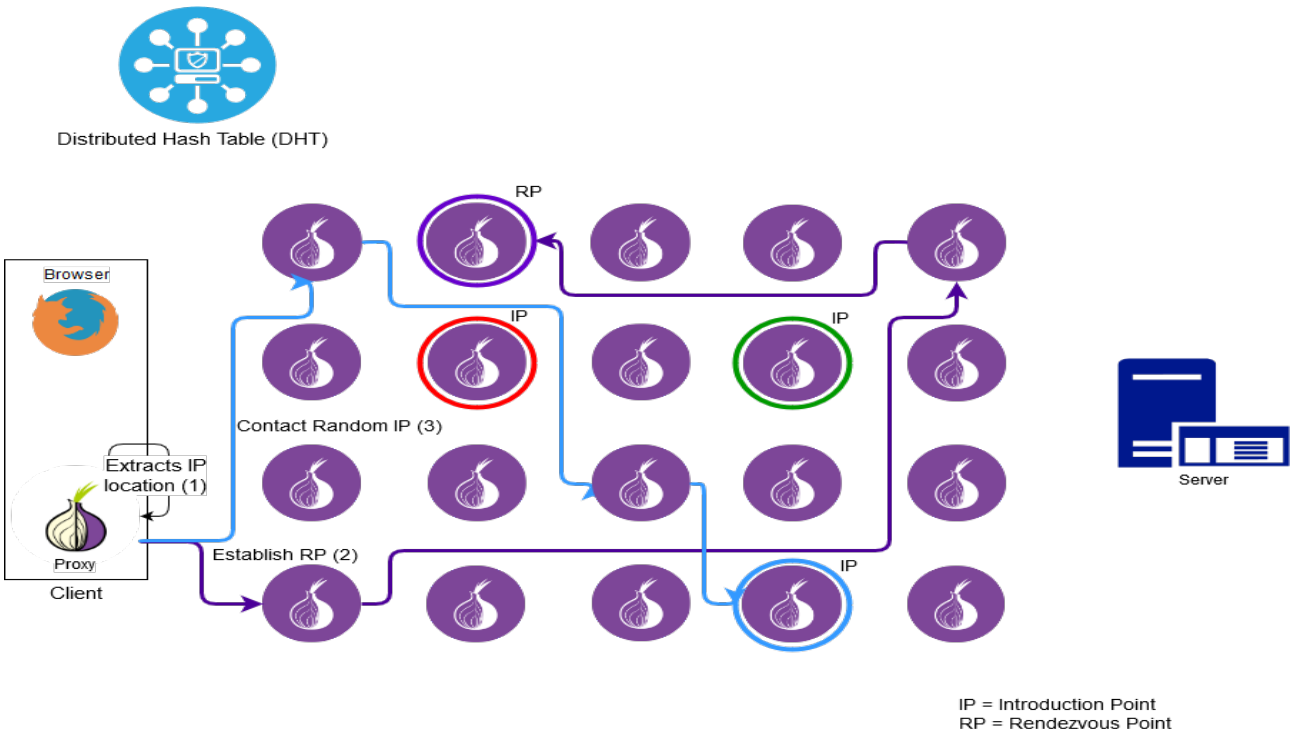


Figure 3: Client retrieving the descriptor and letting the service know where to connect

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 [14].

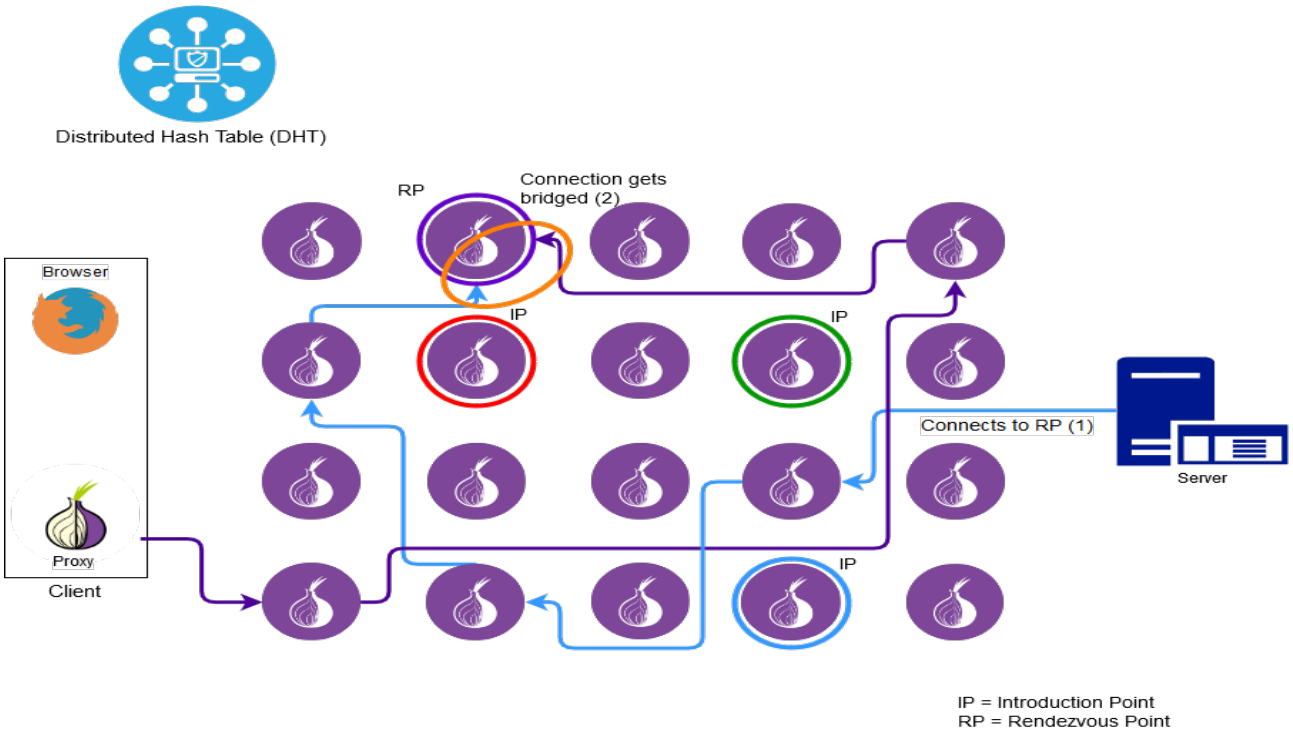


Figure 4: Service and client connecting to the RP and having their independent connections bridged together

# 3 Literature review

In the past, there has been some research done in this specific area 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, deanonymization**. They separate the research into logical sections where first they attempt to retrieve the onion links. Once they managed to retrieve them they correlate it to requests received by clients to statistically analyze and measure the popularity of each service. They then follow with research into other techniques of deanonymization of traffic in the tor network. They also look into generating server fingerprints to match the holes around a service publication. This enables an attacker to get all descriptor publications to his own servers, which in turn enable DOS attacks by denying answers to clients requesting for the specific service [1].

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.

One year after, there was a collaboration between the authors of the paper mentioned above with another author which make extensive use of the 2013 paper to research a technique called shadowing. This technique makes use of the tor network limitation of only 2 nodes per IP and the fact that it keeps track of all extra nodes in case the active ones go down. With this, it is possible to make the active nodes drop out of the consensus and the shadow nodes come forward with the accounted runtime as a shadow, getting that way all of the flags they would have gotten if they were active, without waiting when they turn active [2].

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** which brushes on how hidden service works and what kind of attacks are known to be viable at the time. Also, it mentions the at the time current version of the descriptors (v0) and about a proposal for a new version (v2). It does not mention in any way how to extract onion links but instead mentions a method proposed to defeat denial of service attacks on hidden services 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 is conduction a denial of service [10].

There were some newer papers that use some of the information of these papers mentioned above but nothing that adds to the extraction of onion links and 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

The hardware used to conduct this project can be found in the 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 fist thing done was setting up both servers and making them secure. This involved updating the system and setting up iptables to deny any incoming except for ssh, the tor ports 443 and 9030, and the loopback interface. The next step was to set-up the tor software on both servers. To do this the tor repository had to be added to each server. Once the files were in place the software could be installed and updated as any other package in the system by using the package manager "APT". Once the packages were installed they were configured with the configurations that can be found in appendix A. The software was then started using the system services manager "Systemd". Once this was done nyx was started to monitor tor using its control port (9051). An overview of the nyx interface can be seen in figure 5 below:


Figure 5: Nyx interface [11]

The next step was to wait for the required flags, HSDir and Stable, to be delegated to the server. The HSDir flag required 4 days to obtain it but first, the server needed to obtain the stable flag which takes 5 days running on average. Once the stable flag is obtained for the first time the HSDIR also gets delegated right away. Once HSDir flag was obtained Hidden services are able to publish descriptors to the servers. This enables the extraction of the descriptors through memory dumps.

## 4.2 Approaches

It was assessed that to retrieved onion addresses there are 4 possible ways. These are internet scraping, address bruteforcing, network sniffing, and memory extraction. The methods that were chosen for this research were network sniffing and memory extraction.

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 although a valid method to find hidden services and entirely possible due to the small size of v2 addresses (16 characters links). It is time-consuming and would require the fetching of the descriptor to see if it is valid.

The network sniffing was concluded to also not be possible to retrieve since the v0 protocol was deprecated in favour of the v2 protocol. This is due to on the v0 protocol requests done to the DHT being done using the onion address itself as opposed to an ID as it is accomplished in the v2 protocol. Despite this, it was still deemed useful for cross-referencing requests with know onion addresses for statistical analysis of the hidden service usage.

The memory extraction is a sure 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.

### 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 generalized terms for extraction control. Patterns such as: "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 memory sniffing method did not yield any descriptors either from the publication from hidden service providers or retrieval requests from clients. Although this was the case, the control pattern "GET" did capture outgoing requests and responses by the server for general tor resources.

The problem with the network extraction can be attributed to one or more reasons. First, it could do to caching of the descriptor by the client [8]. Although 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 which prompts for a new set of servers. They get published the first time and only need republishing within the 24 hours if any introduction points are added, removed, or changed [2]. Although this is the case, it is strange that no new publications were captured despite the period the network was being sniffed had been brief.

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 [13]. 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.

Lastly, the least plausible reason, the specification describes the descriptor requests and publications as HTTP get and post requests [13]. It could be that the specification has changed and not updated, which in turn would turn the search patterns used ineffective.

Even if the capturing had been successful, the captured information would not be directly possible to extract onion links. This is due to the clients using, in v2 and v3 descriptors, the ID that can only be calculated from the onion address but not back since it has a hashing mechanism step. Although this is the case it is possible to use the captured ID to request the descriptor from another responsible server. 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 [3].

### 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 non library memory sections and only 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 6 below:

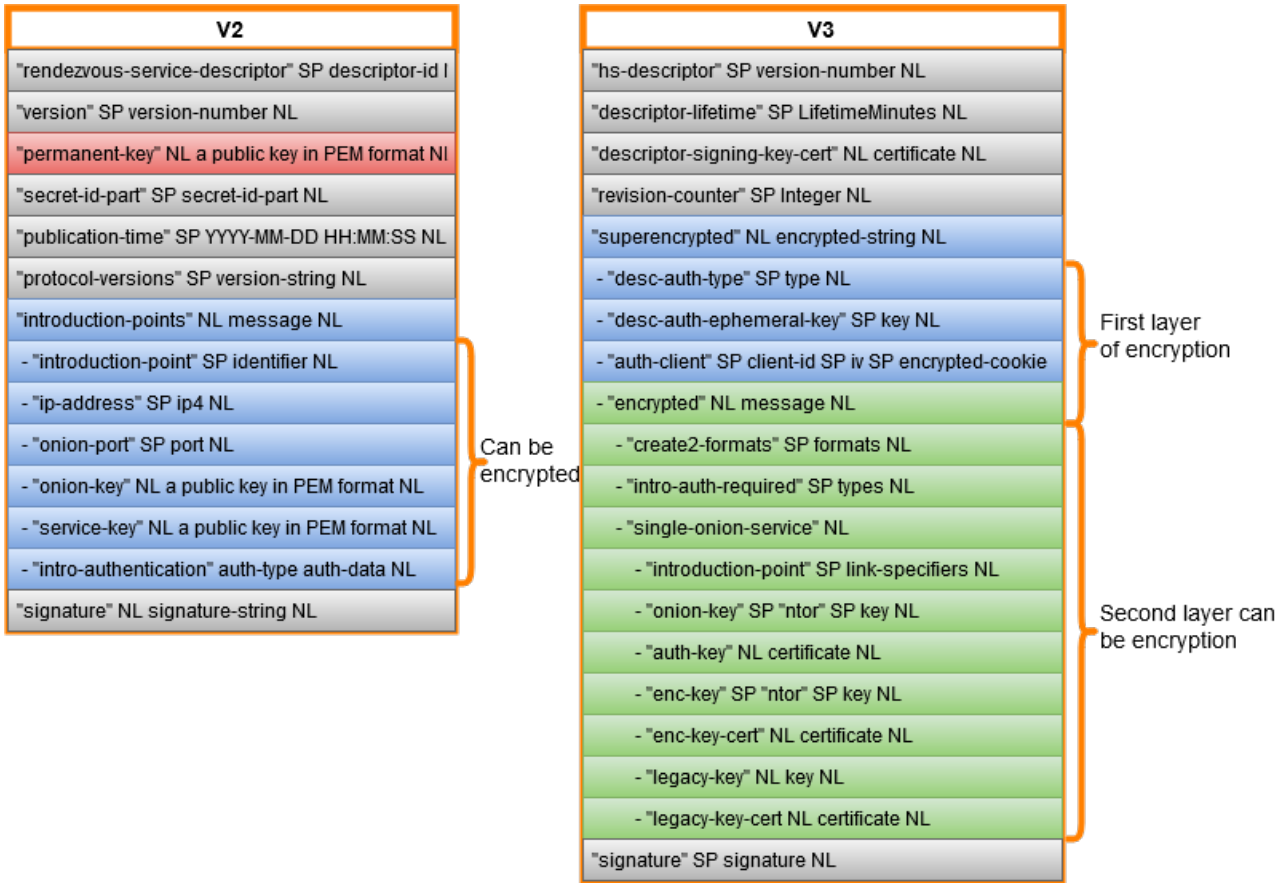


Figure 6: Descriptor structure for both versions of the descriptor in the current specification

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 then passes 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 was very successful. Descriptors can be reliably extracted by following the structure present in the specification. From v2 descriptors, it could be extracted onion addresses without any issues. From v3 it was not possible to extract any onion address.

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 6 where the v2 has the permanent-key (red field). On the other hand, the v3 adds more layers of encryption with only metadata necessary for identification of the descriptor being left unencrypted. Furthermore, v3 does not contain the public key even in the encrypted section [13]. 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 GitHub using the link found in appendix B. 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 7.

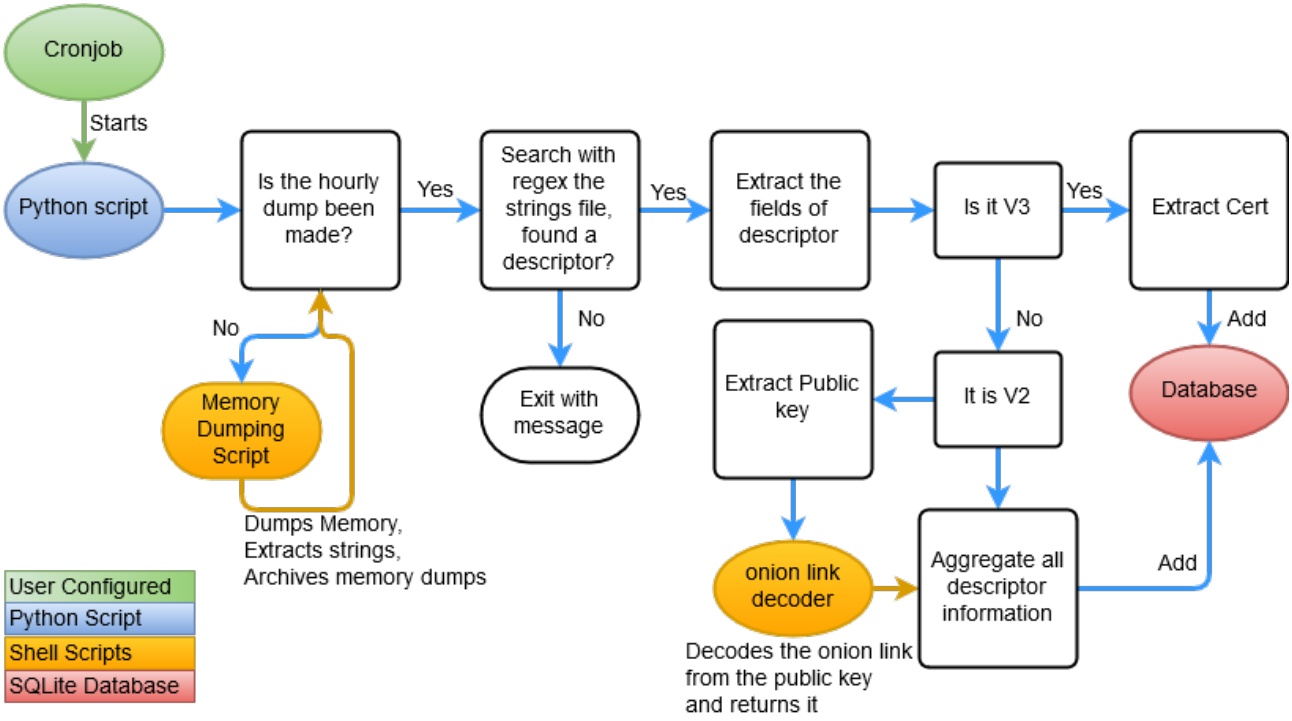


Figure 7: Process diagram of the python framework that extracts and processes descriptors directly from an HSDir process memory

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.

The automated tool was set to run as an hourly cronjob and the result of running it for 5 days can be seen in figure 8

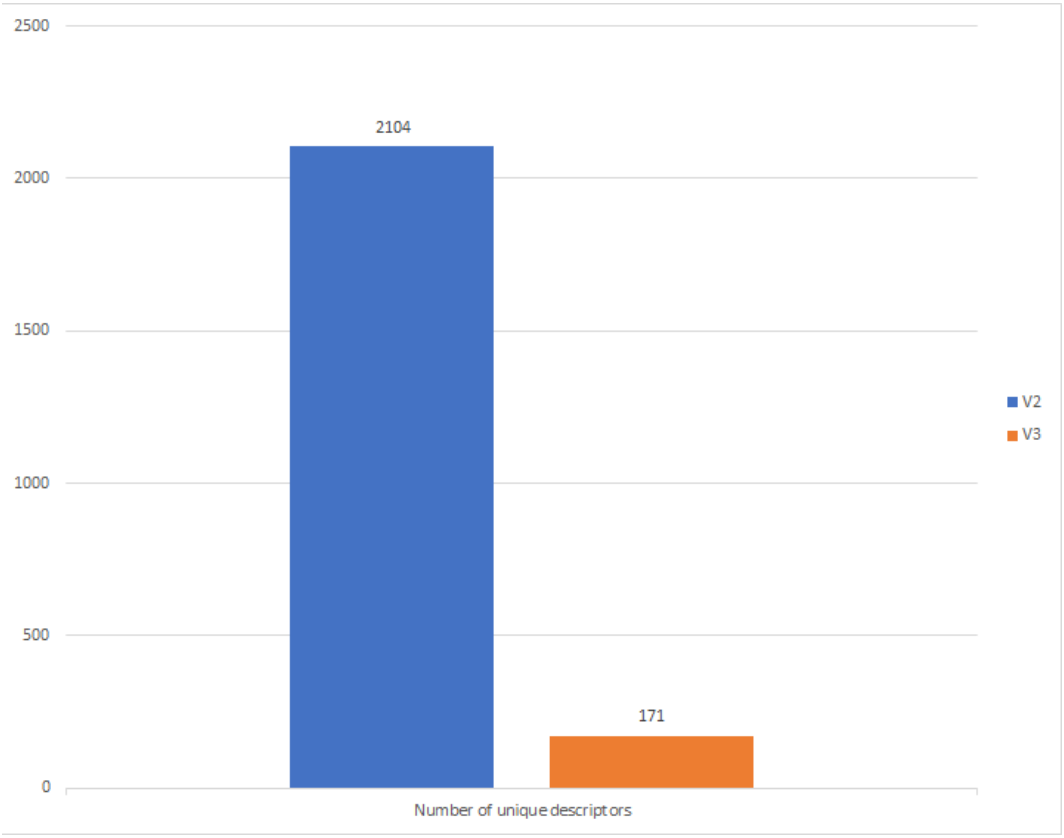


Figure 8: Comparison of descriptor captures between v2 and v3 on hourly 5 day runs

It can be seen that in 5 days the tool captured 171 v3 descriptors and 2104 v2 (12.3 x more than v3). This shows that there has not been a big adoption of the v3 protocol.

Both servers were running the tool and writing to the same database. The data in the figure 8 shows mostly captures from the OS3 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 database. 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 of any data that had been able to go through. This problem could have been easily mitigated with optimization 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 further shows how much it can capture with the few resources it has.

The cost of this experiment was 2 servers for less than 26€, capturing for 5 days. When comparing with 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 [12].

# 5 Countermeasures

Despite the research revolving around the finding of illegal content services and monitor said services. It is against the core principals 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. This might not be an easy accomplishment due to the number of services that still rely on the v2 protocol. Another solution would be the service providers generating v3 services as opposed to v2. The problem with the usage of v3 is that 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 [17] and scallion [16]. Also, thinking of the users having to remember or keep a record of 16 characters is better than 56. When all of these are not possible, it is recommended that hidden services make use of the authentication of its users with a password/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 as access to the descriptors it would not be able to use it as a stepping stone to launch attacks or gather intelligence on the hidden service.

# 6 Future Work

With some more time, more work could be put into the network extraction. 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 there is really no way of recovering the onion link. Furthermore, it would be interesting to see if it is possible to extract information from the service providers servers as well as analyse how many hidden services actually use the client authentication methods. Moreover, assessing if the authentication methods can be brute-forced or exploited in any way could be interesting. Also, 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 [9].

# 7 Conclusion

The need for privacy and anonymity was rewarded with the creation of overlay networks such as the tor network. The Tor network is still being actively developed and new and improved features being added/changed. Some of these features also provide anonymity service providers alike. This enabled users to start hosting illegitimate content alongside legitimate one. This raises the need to secure legitimate content and monitor/block illegitimate one. One way to find these hidden services to assess the legitimacy of the content is through their onion address which not always is shared so 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 a large number of onion addresses through memory extraction of servers belonging to the Distributed Hash Table. Furthermore, it shows it can be done with cheap reasources and a small amount of time. The findings of this research can be easily used as a stepping stone for further attacks and/or intelligence extraction from the Tor Networks users. For this reason, it is imperative that the newer protocol received a bigger adoption and/or the security features of the older ones be used to keep the users protected from malicious parties. This research shows the necessity for improvement in this area to keep for example users from oppressed environments secure and anonymous.

# References


[1] 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.

[2] 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.

[3] 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.

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

[5] 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.

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

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

[8] 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.

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

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

[11] *Nyx: Command-line monitor for Tor*. URL: https://nyx.torproject.org/#home.

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

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

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

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

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

[17] *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 3: 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 4: Namecheap Tor Configuration

## B Github Project Link

https://github.com/JCMarques15/tor-hs-fetcher