Using HSDirs for TOR network analysis

Ashwin Kumar

January 2020

Rotation Supervisor: Roch Guerin

Duration of the Rotation: 2 Jan - 28 Feb

1 Tor

Tor is a routing system based on onion routing, used to provide anonymity and privacy to its users. Onion routing allows for a circuit consisting of multiple routers, where the sender encrypts the message with multiple layers of encryption corresponding to each router in the path, and each router strips away its layer of encryption while relaying the message to the next, until eventually an unencrypted message reaches the destination. Tor exploits this architecture by making one circuit each from the two communicating parties to a common rendezvous point. The message sent can be further encrypted by a shared key between the two parties. This system ensures the privacy and anonymity of both the client and the server, unlike traditional onion encryption, which protects only the sender.

To access a hidden service on tor, one requires an onion name, which consists of a cryptographic hash of the public key of the hidden service appended with '.onion' at the end. The tor network does not have any central directory housing all domain names, or a DNS. Thus, client who need to access hidden services need to find out about their domain names from public listings or from out-of-band channels. This allows a hidden service to maintain secrecy while operating for a select few people who know about it.

This highly anonymized nature of tor has also led to it being a host to various illegal activities. It might be of importance to trace such services and stop their activities, but the lack of DNS or any central authority makes it especially hard to do so. Hence, many people have tried to find ways to populate lists of .onion names to crawl through and categorize their content. With the increasing use of v3 services, we seek to see whether it is still possible to do the same, and to get an estimate on the number of active v3 services.

2 Current implementation: v3

Most of the literature analyzing and attacking tor is for the version 2 (v2) specification, while tor has come out with a new and more robust v3 implementation. In its current version, tor protocols for Hidden Service Directories are significantly different from the v2 protocols. v2 directories and services suffered from a lot of shortcomings like

- Possibility of DoS attacks by impersonating responsible HSDirs. This was possible because the position of each service on the DHT was deterministic
- Possibility of uncovering hidden service names using snooping HSDir's by looking at the unencrypted data they store.
- Shorter names making it easier to do partial brute-force attacks to fool users
- Older cryptographic functions

The v3 protocol (1) updates the security measures and makes it harder to trace or de-anonymize services and users. This version mandates the use of longer 56-character onion names which encode the ed25519 key of the onion services, as compared to the 16-character RSA SHA1 hash used for v2.

One of the most important changes made in v3 is in the way descriptors are stored on a Hidden Service Directory (HSDir). Any client wishing to query for a descriptor requires knowledge of the blinded signing key, and further needs to know the unblinded public key of the service in order to decrypt the contents. This means that information about any hidden service is unavailable to agents who are unaware of its public key. This prevents collection of v3 service names by agents using snooping HSDir's, as they can only observe the encrypted data, and there is no way to tie this information to a service they do not already know about. The blinded signing key can be derived for any given time period by any client using the public identity key, which is communicated out of band.

Next, this version also implements a daily Shared Random Value which is used to determine the position of each relay and service on the DHT. Since it is impossible to predict where any service will be on the circle before this value is published, it is hard for anyone to impersonate responsible HSDir's for a service, mitigating the possibility of a Denial of Service attack. Further, since this value is updated daily, it doesn't give a malicious actor time to create a new relay which can be inserted at that spot, as getting the HSDir flag takes at least 96 hours.

3 Getting Hidden service descriptors

Process:
Whenever a client wishes to connect to a Hidden service, but does not have the descriptors, they use a combination of the blinded public key (derived from the public key of the hidden service) and the time period to determine which relays(HSDir's) are storing the descriptors to find the Introduction Points to set up a connection to that service (There is a method for authorized clients only to connect as well). The descriptors are signed by the blinded public key, hence the HSDir's cannot decrypt them to find out about new onion addresses. Clients, who have had the public keys communicated to them out-of-band, can privately use that knowledge to derive the blinded keys and correspondingly locate the service. The client will need the unblinded public key of the service to decrypt the body of the descriptor. (Each introduction point also has its own authentication key stored in the HS Descriptor)

The client then proceeds to select a rendezvous point and communicate a nonce to the introduction point, which contacts the server and establishes a connection through the rendezvous point using the nonce.

The client requires the v3-hidden service name in order to initiate this connection. It is the service's long term master identity key. This is encoded as a hostname by encoding the entire key in Base 32, including a version byte and a checksum, and then appending the string ".onion" at the end. The result is a 56-character domain name.

List of keys:

Master (hidden service) identity key – A master signing keypair used as the identity for a hidden service. This key is long term and not used on its own to sign anything; it is only used to generate blinded signing keys. The public key is encoded in the ".onion" address as described.

Blinded signing key – A keypair derived from the identity key, used to sign descriptor signing keys. It changes periodically for each service. Clients who know a 'credential' consisting of the service's public identity key and an optional secret can derive the public blinded identity key for a service. **This key is used as an index in the DHT-like structure of the directory system** Candidate methods for deriving this blinded key are present in the appendix of tor-spev-v3.

Each authorized client possesses:an x25519 keypair used to compute decryption keys that allow the client to decrypt the hidden service descriptor and maybe a client authorization keypair

Generating and Publishing Hidden Service descriptors:

Blinded signing keys (private) are generated every time period an used to sign descriptors. Clients use a different blinded public keys to fetch this information. They both also have a *subcredential* for each period which is needed to decrypt the HS descriptors. Unlike the credential, it changes each period.

$$credential = H("credential"|public - identity - key)$$
$$subcredential = H("subcredential"|credential|blinded - public - key)$$

HSDir locations are no longer predictable even if the location of the service can be predicted because the DHT table is indexed by the shared random value combined with the HSDir's public identity keys. Which Tor servers hosts a hidden service depends on:

- the current time period,
- the daily subcredential,
- the hidden service directories' public keys,
- a shared random value that changes in each time period,
- a set of network-wide networkstatus consensus parameters.

Time periods are 1440 minutes long by default(one day), starting from Jan 1, 1970, adjusted to 12:00 pm UTC. Therefore, HSDir's are reallocated everyday at noon[1]. Every random interval between 60-120 minutes, each hidden service publishes its HS descriptor to the HSDir responsible for that time period.

Hidden services also have to keep publishing the previous copy of their descriptors to the older HSDir's for clients who have the previous consensus. They also upload new descriptors to the next set of HSDirs before the next time period. So, multiple copies of the same service's descriptors can exist simultaneously, each answering to a different consensus.

The Shared Random Values are uploaded at midnight for the next time period at noon. Every consensus contains two SRV's, out of which one is the previous time period's.

Forming the HSDir Ring:
There are replicas of descriptors stored in 2 sets of 4 HSDir's each by default. 4 is for redundancy, and that will hopefully reduce the number of HSDirs we will need to run to get an accurate statistical estimate. Clients still read the next 3 HSDirs, though.

[1]The staggered nature of the shared random value and time period mean that each descriptor needs to be published in two places

To get the index for each descriptor replica on the HSDir ring, the following hash is used once the blinded public key has been computed:

```
    for replicanum in 1...hsdir_n_replicas:
.             hs_index(replicanum) = H(”store-at-idx” |
.                           blinded_public_key |
.                           INT_8(replicanum) |
.                           INT_8(period_length) |
.                           INT_8(period_num) )
```

Following this, for each HSDir in the current consensus, their index is calculated as:

```
.        hsdir_index(node) = H(”node-idx” | node_identity |
.                           share_random_value |
.                           INT_8(period_num) |
.                           INT_8(period_length) )
```

where shared_random_value is the shared value generated by the authorities, and node_identity is the ed25519 identity key of the node.

For each replica, the descriptors are uploaded to the first 4 HSDirs whose indices follow hs_index(replicanum).

Staggered SRV and time period
The calculation of the SRV (shared random value) is staggered by 12 hours with the time period
SRV : 00:00
Time Periods: 12:00

This creates 2 kinds of zones, 00:00 - 12:00 and 12:00 to 00:00 The first zone has a newer SRV, but older time period, the second has a newer time period but older SRV

In each zone, the first descriptor is uploaded using the previous zone's SRV and TP

The second descriptor is slightly different. In zone 1, it uses the next time period (after 12:00) and current SRV (for clients which have a newer consensus) In zone 2, it uses curent time period and current SRV for clients with up to date consensus. (Are these two descriptors the same as the replicas, or additional? Affects final calculation)

4 Process pipeline

Our goal is to set up an environment where we can attempt to harvest descriptors from a tor relay we own, which has obtained the HSDir flag. Then, we attempt to understand the structure and contents of the descriptors and attempt to track them across different uplaods. We also attempt to get an estimate on the number of active v3 services.

The process pipeline is as follows:

- Start a tor relay and obtain the HSDir flag
- Get the descriptor information stored on the relay, and create periodic dumps of the information.
- Process the dumps to get descriptor counts and common files across different times.
- Analyze the processed information for patterns and trends.

To set up a tor relay, we used a VPS instance from hostworld (5) with 30GB storage and 2GB RAM. It used an unmetered connection, which was beneficial for us because a tor relay demands high bandwidth. Indeed, over the course of operating for just a week, our relay saw over 16 TB of traffic.

We used Ubuntu 18.04 as the OS, and ran Tor 0.4.2.6. It took, on average, 5 days of continuous operation to obtain the Stable and HSDir flags after starting the relay. We used Nyx (4) as a command line tool for getting real time data and debugging.

Getting the descriptor information is not straightforward, as tor does not store that information in a file. To get the descriptors, we had to extract memory mappings of the tor process and create memory dumps by scraping the process. These memory dumps also turned out to be huge (of the order of 500MB), and thus we used the strings utility to remove any non-ASCII characters, and removed the dumps from memory. Then, these dumps were processed using python to extract the descriptors using the meta-format provided in the tor v3 specifications document. A set of cronjobs was used to execute all these tasks hourly. Another separate python script was used to aggregate the information present across all separate dumps and compress them into a single file with just the total counts and number of common descriptors across different files.

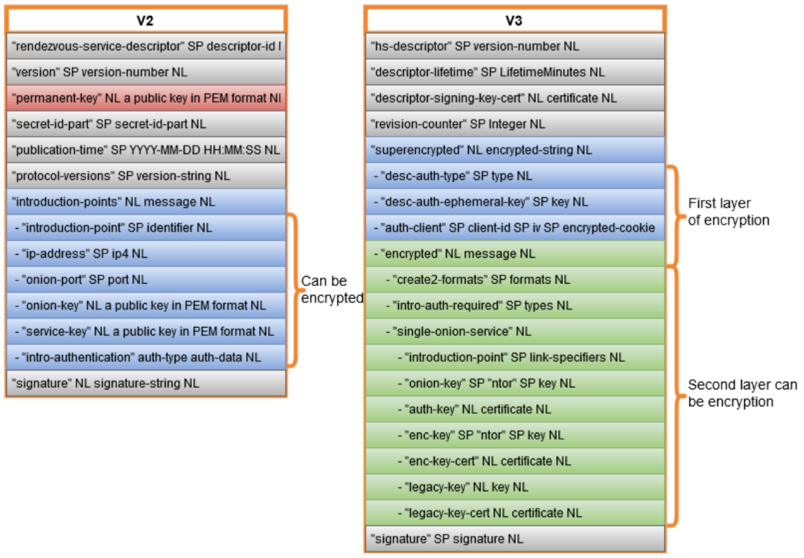
V2
"rendezvous-service-descriptor" SP descriptor-id I
"version" SP version-number NL
"permanent-key" NL a public key in PEM format NI
"secret-id-part" SP secret-id-part NL
"publication-time" SP YYYY-MM-DD HH:MM:SS NL
"protocol-versions" SP version-string NL
"introduction-points" NL message NL
- "introduction-point" 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
Can be encrypted
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
First layer of encryption
Second layer can be encryption

Figure 1: Descriptor meta-format for v2 and v3 (3)

5 Results

Since even the compressed data was large, a visualization was created for this data, to help in interpreting the results. As shown in the figures, one area plot is drawn for each file, and the overlapping regions show how many descriptors are common between the two files. The circles show which points have been samples, and the graph is interpolated between these points.

For the v2 descriptors, the ”permanent key” field was used to find similar entries across files, and for v3, the ”descriptor-signing-key-cert” was used. It is to be noted that while the permanent key remains same for all future re-uploads of a v2 descriptor, the same is not true for v3 descriptors.

As can be seen from the graphs, v2 descriptors tend to stay in the memory for a long period of time, very rarely being dropped (this includes re-uploads of the same descriptor at a different time), while v3 descriptors only stay in the descriptor for a few hours usually. v3 descriptors cannot be tracked across re-uploads as there is no identifying information in the unencrypted section of the descriptor. We observe a spike in the number of v3 descriptors everyday at 18:00.

Now, to estimate the total number of v3 services we go through the following process. First, using the collected data as a statistical estimate, we obtain the average number of v3 descriptors stored in a relay at any time. Then, using data from Tor Metrics (2), we find out the total number of HSDirs online, Y.

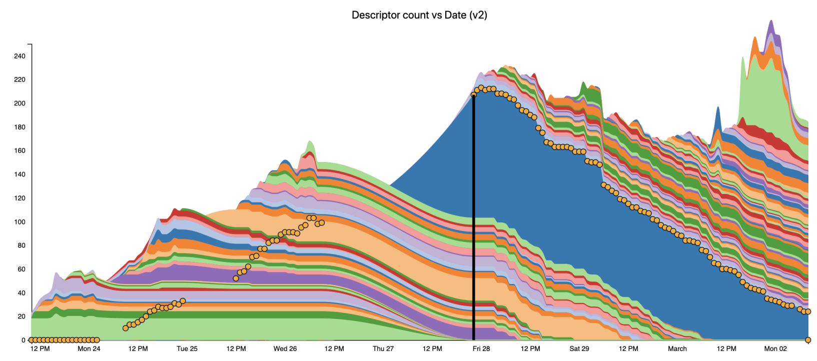

Figure 2: v2 Descriptor counts

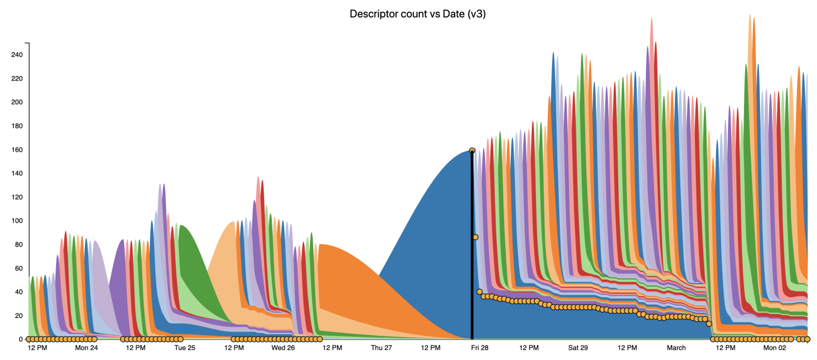

Figure 3: v3 Descriptor counts

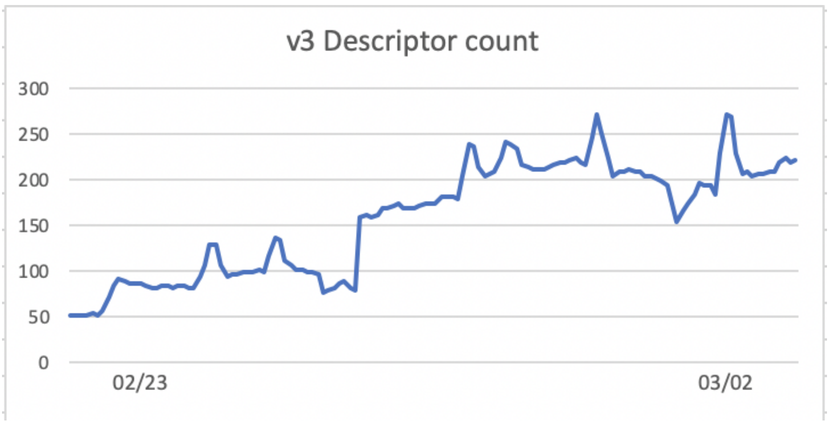

Figure 4: v3 total descriptors

Each hidden service stores its descriptors in *replicanum* replicas and across 4 HSDirs for each replica. Further, each descriptor is uploaded to 3 sets of HSDir's for different time periods. Hence,

$$X = \frac{V3_\mu * Y}{replicanum * 4 * 2}$$

where $V3_\mu$ is the average count of v3 descriptors.
Plugging in the sample values we have, we get

$$X = \frac{158 * 3768}{2 * 4 * 2}$$

$$X = 37209$$

Thus, the data that we have would seem to suggest that there are around 30,000 active v3 services at any time. This number is expected to be a little higher, as we see the number of v3 services stored on our relay increase over time.

References

[1] rend-spec-v3.txt - torspec - Tor's protocol specifications. Accessed February 15, 2020. https://gitweb.torproject.org/torspec.git/tree/rend-spec-v3.txt.

[2] Tor Metrics. Accessed March 1, 2020. https://metrics.torproject.org/.

[3] Marques, João. " Tor: Hidden Service Intelligence Extraction,". https://www.delaat.net/rp/2017-2018/p98/report.pdf

[4] "Nyx." Nyx. Accessed February 10, 2020. https://nyx.torproject.org/.

[5] "Welcome to the World of Hosting!" hostworld. Accessed February 10, 2020. https://hostworld.uk/.