CSEC 793 Capstone in Computing Security
Project Report

Obsucre Communication using Tor Relay and Hidden Service

April 23, 2019

Priyank Jani
Department of Computing Security
College of Computing and Information Sciences
Rochester Institute of Technology
`ppj4900@rit.edu`

1 Abstract

Covert/Obscure Communication is a field in Information Security which is being extensively researched and implemented as it has a significant impact on todayâs cyber age. With the advent of quantum computing, social networking, censorship and extensive surveillance; it is becoming extremely arduous to preserve oneâs privacy or maintain consistent anonymity over the web. The idea of distributing information covertly has been practiced since ancient times. Ancient Egyptians communicated covertly using the hieroglyphic script, a series of symbols representing a message. The Greeks used âhidden writingâ which is the derivative of steganography.

Even though it has become rudimentary knowledge that extensive surveillance leads to Internet users being monitored, the Web having a distributed architecture becomes an important medium which can be utilized to build obscure channels for communicating covertly. The use of Internet infrastructure for building such a channel can be made to hide information as well as communicate with enhanced privacy. Hence, the Internet can be used to establish such channels to hide information and communicate privately between two endpoints. Even though a major motivation for using such channels could be for crime, hacking, espionage among others; covert channels find extensive applications in military communications, privacy enhancing technologies, censorship circumvention and cyber offense.

The purpose of this capstone project is to successfully establish an obscure communication system over the Onion Router network (Tor) using Tor Relays and Tor Hidden Services. Tor is a privacy enhancing technology providing anonymity to its users. Hence establishing an obscure channel using Tor infrastructure adds a strong layer of anonymity and abstraction to the channel making the process of communication more difficult to detect.

This capstone project is unique, in the sense that, it has very intelligent design and its successful implementation explores an undertaking not researched before.

2 Introduction

- The goal of this project :

  This capstone aims to establish an obscure/covert communication system between two entities Alice and Bob using the Onion Router (Tor) infrastructure. Onion URLs for Hidden Services have to be made available to the Tor users out of band. This capstone aims to obscure an Hidden Service's onion URL by encoding it and including it in the Tor network's meta-data.

- The rationale of the project :

  This capstone has been designed to use the anonymity provided by the Tor network and build an extra layer of obscurity over the anonymity layer to enable hidden/covert communication between two entities.
  A Threat Model that considers malicious Hidden Service Directory Servers, critical to Hidden Service Operations, has been developed and is necessary to be mitigated for successful implementation of the obscure communication system to successfully function.

- The importance of the project :

  Upon careful and successful deployment, this obscure communication system can be used for covert activities to be used for defense projects, to communicate within the Tor infrastructure and even military operations overseas.
  Any communication system between base and remote military locations can modify this communication system and deploy it for obscure communications as this system is built upon the anonymity of the Tor network.

- Main contributions to the project :

  Designing the entire obscure communication system using Tor infrastructure which has never been undertaken before.
  Hosting a live Tor Relay and a live Tor Hidden Service on the internet.
  Writing scripts for obscuring the access to the Hidden Service.
  Testing the obscure communication system as it is live on the internet.

3 Literature Review

A storage channel is a channel where shared storage media is used to encode and transfer information. A timing channel used as a means of communication relies on the time between events to encode and transmit information, instead of shared storage media. Non-stored and time independent channels are known as behavior channels. [1]

Gimbi et. al.[2] present a technique that uses transport layer source ports as a channel for covert communication. The work also discusses various encoding schemes used with the channel communication as they provide data integrity and obfuscation from auditors. Data is being transmitted in the header fields of TCP/IP packets using port numbers. As far as communication using standard port numbers are concerned, a usable ephemeral range is provided by the Internet Assigned Numbers Authority (IANA) of 49152 through 65535 for a total of 16384 available ports. Layer four protocols provide flexibility in source port selection. By manipulating the source port, upto 16 bits of information can be transmitted from the source to the destination as all that is being modified is the content of the mandatory static length field which can be easily piggybacked on top of legitimate traffic. The channel is made more robust by using the difference between two consecutive source ports for transmitting the ASCII character represented by the difference. The covert channel is limited by some inherent shortcomings. Its use can be limited by the deployment of Network Address Translation as many NAT implementations modify the socket hence changing the port numbers. If the source is behind the NAT box, it fails to communicate with destination. Also, proxy servers change the socket pair, hence ruining the channel. There are also drawbacks in the legitimate implementation, in that, if another unrelated process makes use of an ephemeral port, the particular port will be locked from other processes till the TIME-WAIT timer, built into TCP to ensure that the socket can still properly handle traffic arriving late from a closed connection, expires. Also, if the source and the destination communicate outside of the channel process, the destination would interpret the outside communication as legitimate traffic. This problem is effectively eliminated by the intelligent encoding mechanisms described in [3].

Anthony et. al. [4] describe a novel behavior based covert channel that utilizes the database update mechanisms of antivirus software. An anti-virus software gets updates for signatures to detect malware which are frequent, hence modifying and adding to their databases. The Anti-Virus covert channel is behavior based and used to communicate without being flagged by the security implementations of an enterprise. The sender creates custom signatures to be exported into an AntiVirus database distribution point. The receiver downloads and updates his Anti-Virus from the same distribution point and interpretation for the covert message is done on a directory of files shared with the sender. ICMP Protocol is used widely for network and system troubleshooting. The protocol has several codes for correctly diagnosing traffic flow and network problems. Covert channels using the ICMP protocol are classified as storage channels. Several fields in ICMP messages are unused and are utilized for covert communication. Data fields and payloads within certain ICMP messages are used covertly resulting in quick implementations and simple channel setup/teardown. ICMP has a very small footprint as it is a lesser standard for communication than TCP or UDP. But the use of the ICMP protocol as a covert channel comes with drawbacks as several countermeasures have been developed over the years to detect and thwart covert communications using ICMP.

Stokes et. al. [5] describe various countermeasures used against ICMP covert channels and test the resiliency of ICMP covert communication deploying these countermeasures. Blocking all ICMP traffic may not be viable because of loss of network troubleshooting capabilities. Blocking ICMP traffic at the firewall level also does not solve the problem as hosts in the internal network could still use it. Blocking specific ICMP messages could be thwarted using fuzzing techniques at the source and destination of the covert channel which modify the ICMP messages hence surpassing the traffic restrictions. If ICMP messages which have extensive data fields or large payloads are blocked, the channel operators could fragment the ICMP messages to smaller packets which would reassemble once reached inside the destination network. The researchers test the resiliency of ICMP-Chat and Ping Tunnel applications placing a Checkpoint Firewall and a Snort Intrusion Detection System. Both applications were successful in covert communication proving that ICMP based covert channels can subvert modern security appliances if ICMP traffic is permitted.

The goal of this capstone is to build a covert communication system using Tor infrastructure and metadata. The system would use the obscurity and privacy provided by Onion Services which are used for responder anonymity and the Onion Relay ”Nickname” descriptor field. It is common to call an Onion Service as a Hidden Service.

Onion services are very important for people who want to host content without the fear of being targeted, arrested or forced to close shop. This survey tends to describe the complete operation of Onion services taking the reference of previous work centered around Onion services.

A Server is created behind an Onion Proxy running on the system. The address for the Onion Service is available for the users to access it out of band. The Hidden Service's Onion Proxy chooses a random node in the Tor network to serve as an Introduction Point and creates a circuit to the Introduction Point.

After key exchange and agreeing to proceed further, the Hidden Service creates a signed descriptor which contains a timestamp, information about the Introduction Point, and its public key. The Hidden Service then computes a descriptor-id based on the public key hash and the validity duration. The descriptor-id is then published to the Hidden

Service Directories. The Hidden Service Directories are a group of Onion Routers in the Tor network that are flagged as having the HSDir (Hidden Service Directory) Flag set and are sharing the hash ring of all the Hidden Services in the Tor network.

The client that wants to connect/communicate to the Hidden Service shall now fetch the descriptor from the Hidden Service Directories. The steps for this are as follows: first, the client decides a Rendezvous Point and communicates information about this chosen point to the Introduction Point it fetched from the descriptor. There are more than one Introduction Points hence a single Introduction Point is chosen. Once the Introduction Point agrees to this communication, the client send the address of the Rendezvous Point and a one-time secret via a cookie. All communication is encrypted as these exchanges are taking place in the Tor network. The Introduction Point then relays the information regarding the Rendezvous Point to the Hidden Service and finally, both the client and Hidden Service communicate via the Rendezvous Point securely. [6]

It is very crucial to keep changing the underlying onion service name to avoid detection. There are 9 Directory Authority Servers in the Tor network which maintain a list of all the active Tor relays in the entire Tor network as the Authorities monitor self-advertisements by the Tor relays. The list is maintained by the Directory Authorities and distributed among all the Tor clients. The condition for every relay to be on this list is that the number of Tor relays per IP address should not exceed 2. For a client to host a Hidden Service, its Onion Proxy generates an RSA key pair. A SHA-1 digest of the key is taken, and the first 10 bytes are base-32 encoded to make a .onion URL for the Hidden Service.[7]The essentials for a client to communicate to a Hidden Service are its .onion address, the public key and a list of its Introduction Points. Hence, the Hidden Service generates 2 service descriptors and uploads them to 6 Hidden Service Directories. For the Hidden Service Directory to have an HSDir flag set, it needs to be operational for at least 25 hours.[8]

Onion services are vulnerable to detection and traffic analysis attacks. A common attack is using flow watermarking. This is a brief study of detection and de-anonymization attacks which use flow watermarking in various capacities.

DROPWAT is an algorithm that uses networkâs response of packet loss to its advantage.The adversary (onion service host) would be unaware of the watermark embedded by using DROPWAT. DROPWAT even works when traffic is tunneled through proxies. [9]

Inverse flow watermarking is a unique technique used to locate hidden services by the INFLOW algorithm. Congestion mechanisms have an impact on routing traffic in the Tor network which is used to effect by the INFLOW algorithm.INFLOW detects the source of the onion service by controlling the communication edges i.e. entry and exit traffic and identifying gaps in watermarking. [10]

4 Project Idea

The purpose of this capstone project is to successfully establish obscure/covert communication over the Onion Router network(Tor) using Tor Relays and Tor Hidden Services. Tor is a privacy enhancing technology providing anonymity to its users. Hence establishing obscure communication using Tor infrastructure adds a strong layer of anonymity and abstraction making the process more difficult to detect. This capstone project is unique, in

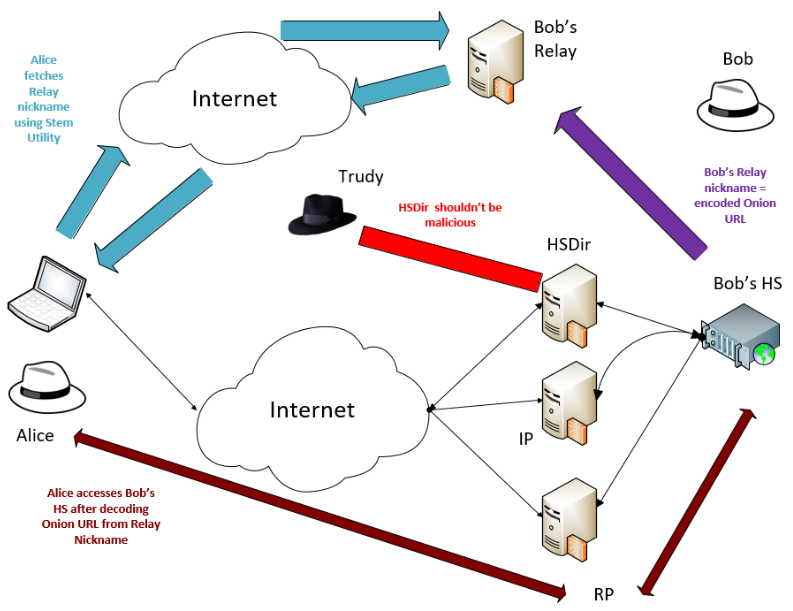

Alice fetches Relay nickname using Stem Utility
Internet
Bob's Relay
Bob
Trudy
HSDir shouldn't be malicious
Bob's Relay nickname = encoded Onion URL
HSDir
Bob's HS
Internet
Alice
IP
Alice accesses Bob's HS after decoding Onion URL from Relay Nickname
RP

Figure 1: Obscure Communication Model

the sense that, it has very intelligent design and its successful implementation explores an undertaking not researched before.

Covert/Obscure Communication is a field in Information Security which is being extensively researched and implemented as it has a significant impact on today's cyber age. With the advent of quantum computing, social networking, censorship and extensive surveillance; it is becoming extremely arduous to preserve one's privacy or maintain consistent anonymity over the web.

The goal of this capstone is to enable obscure/covert communication using Tor Hidden Services. Since Tor Hidden Services can only be accessed using their onion URL, this capstone aims to make the process of obtaining the onion URL obscure and use the Hidden Services to communicate in an obscure/covert fashion. The use of encoding shall make accessing the Hidden Services obscure.

Bob configures his server and readies it to be hosted as a hidden service in the Tor network. The final steps in configuring the hidden service result in a public/private key-pair and an onion URL. Using a pre-developed encoding/decoding scheme, Bob encodes the onion URL.

Now, Bob configures another server to be an active onion Relay in the Tor network and configures its torrc file. For the Relay nickname field in the torrc, Bob includes the encoded onion URL in the field. The Relay becomes a functional part of the Tor consensus

The Relayâs IP address and Control Port number are communicated out-of-band to Alice. Using the Python based Stem Controller utility for Tor, Alice fetches the Nickname of Bobâs Onion Relay as it has become a part of the Tor network consensus. Alice decodes the Nickname to get the onion URL of Bobâs Hidden Service using the same encoding/decoding scheme deployed by Bob. Thus, in the above manner, Alice successfully accesses Bobâs Hidden Service.

Hence, this communication system design totally eliminates the need to publish an onion URL for any hidden service hence adding obscurity to the process of accessing the content of the particular hidden service.

(Next Section on the next page)

5 Project Implementation

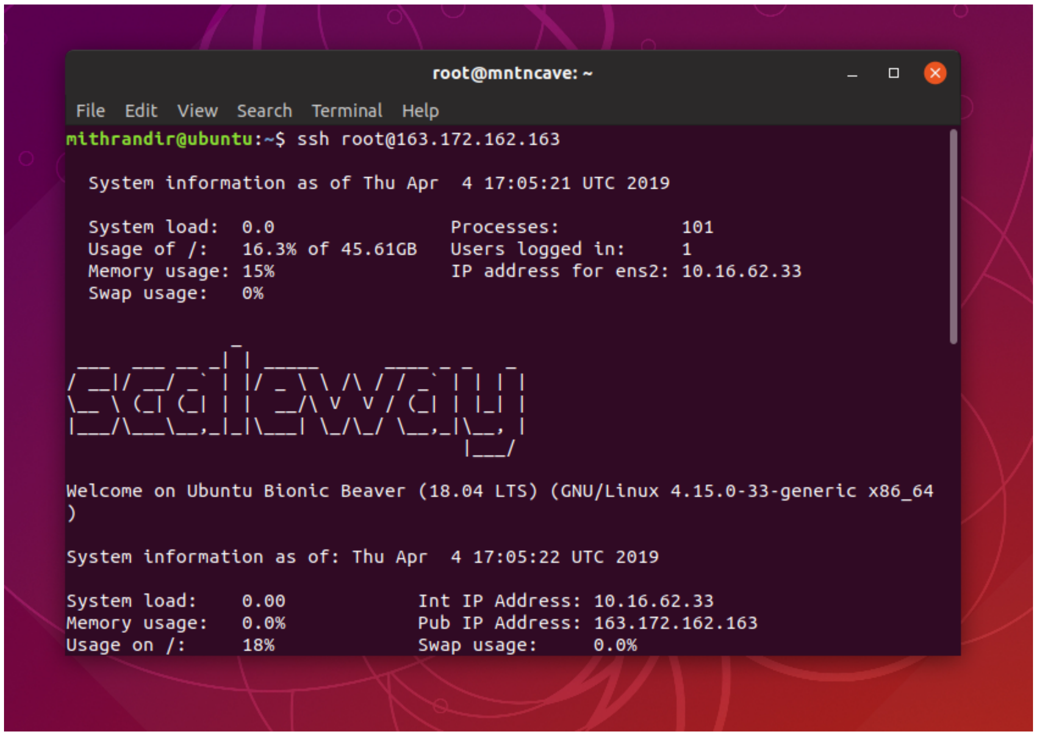
The first step in implementing this obscure communication system is simultaneously configuring a Tor Hidden Service and a Tor Relay.

Both the Hidden Service and the Relay shall be under the control of Bob.

The Tor Hidden Service will host the message to be communicated to Alice by Bob.

The Hidden Service can be any form of Web server, for example, an FTP Server or a normal web-page.

In this case, we are hosting a Simple HTTP Server developed in Python. It is a standard server used for experiments. The server is being hosted on a Bionic Ubuntu server on the cloud service provider Scaleway.



```
mithrandir@ubuntu:~$ ssh root@163.172.162.163

  System information as of Thu Apr  4 17:05:21 UTC 2019

  System load:  0.0               Processes:           101
  Usage of /:   16.3% of 45.61GB  Users logged in:     1
  Memory usage: 15%               IP address for ens2: 10.16.62.33
  Swap usage:   0%


                   _
 ___  ___ __ _| | _____      ____ _ _   _
/ __|/ __/ _` | |/ _ \ \ /\ / / _` | | | |
\__ \ (_| (_| | |  __/\ V  V / (_| | |_| |
|___/\___\__,_|_|\___| \_/\_/ \__,_|\__, |
                                    |___/

Welcome on Ubuntu Bionic Beaver (18.04 LTS) (GNU/Linux 4.15.0-33-generic x86_64
)

System information as of: Thu Apr  4 17:05:22 UTC 2019

System load:    0.00            Int IP Address: 10.16.62.33
Memory usage:   0.0%            Pub IP Address: 163.172.162.163
Usage on /:     18%             Swap usage:     0.0%
```

Figure 2: ssh into the Hidden Service box in the hosted Scaleway cloud service provider

First, we shall install the tor service using the apt utility by using the following command :

sudo apt-get install tor

Now, we shall run the tor utility and start the tor service by using the following commands:

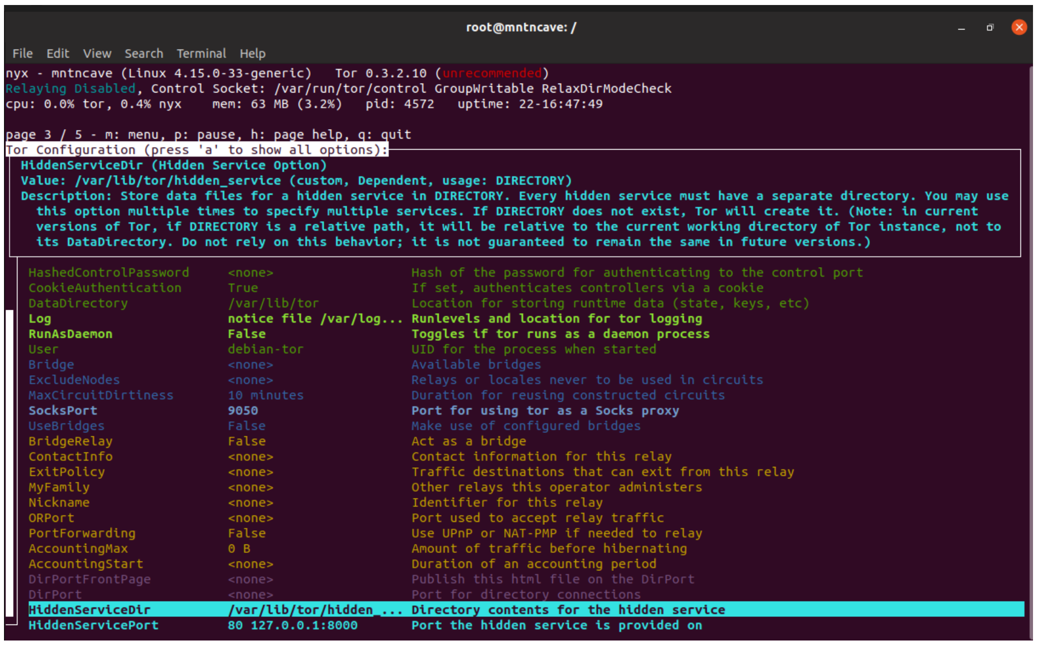
The hidden-service .onion domain has to be created as Tor allows to expose a web-service anonymously. We shall create a directory called mountaincave and setup the web-server.

mkdir mountaincave

cd mountaincave

echo "Hello,Tor!" > index.html

python -m SimpleHTTPServer 8000



```
nyx - mntncave (Linux 4.15.0-33-generic)   Tor 0.3.2.10 (unrecommended)
Relaying Disabled, Control Socket: /var/run/tor/control GroupWritable RelaxDirModeCheck
cpu: 0.0% tor, 0.4% nyx    mem: 63 MB (3.2%)   pid: 4572   uptime: 22-16:47:49

page 3 / 5 - m: menu, p: pause, h: page help, q: quit
Tor Configuration (press 'a' to show all options):
  HiddenServiceDir (Hidden Service Option)
  Value: /var/lib/tor/hidden_service (custom, Dependent, usage: DIRECTORY)
  Description: Store data files for a hidden service in DIRECTORY. Every hidden service must have a separate directory. You may use
    this option multiple times to specify multiple services. If DIRECTORY does not exist, Tor will create it. (Note: in current
    versions of Tor, if DIRECTORY is a relative path, it will be relative to the current working directory of Tor instance, not to
    its DataDirectory. Do not rely on this behavior; it is not guaranteed to remain the same in future versions.)

   HashedControlPassword      <none>                     Hash of the password for authenticating to the control port
   CookieAuthentication       True                       If set, authenticates controllers via a cookie
   DataDirectory              /var/lib/tor               Location for storing runtime data (state, keys, etc)
   Log                        notice file /var/log...    Runlevels and location for tor logging
   RunAsDaemon                False                      Toggles if tor runs as a daemon process
   User                       debian-tor                 UID for the process when started
   Bridge                     <none>                     Available bridges
   ExcludeNodes               <none>                     Relays or locales never to be used in circuits
   MaxCircuitDirtiness        10 minutes                 Duration for reusing constructed circuits
   SocksPort                  9050                       Port for using tor as a Socks proxy
   UseBridges                 False                      Make use of configured bridges
   BridgeRelay                False                      Act as a bridge
   ContactInfo                <none>                     Contact information for this relay
   ExitPolicy                 <none>                     Traffic destinations that can exit from this relay
   MyFamily                   <none>                     Other relays this operator administers
   Nickname                   <none>                     Identifier for this relay
   ORPort                     <none>                     Port used to accept relay traffic
   PortForwarding             False                      Use UPnP or NAT-PMP if needed to relay
   AccountingMax              0 B                        Amount of traffic before hibernating
   AccountingStart            <none>                     Duration of an accounting period
   DirPortFrontPage           <none>                     Publish this html file on the DirPort
   DirPort                    <none>                     Port for directory connections
   HiddenServiceDir           /var/lib/tor/hidden_...    Directory contents for the hidden service
   HiddenServicePort          80 127.0.0.1:8000          Port the hidden service is provided on
```

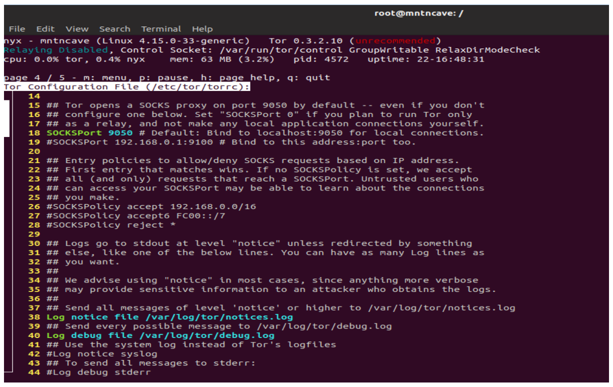
Figure 3: Nyx Output for Hidden Service. The output for this Nyx page shows that Relaying is disabled for this Hidden Service and various metadata related to configuration specifications of this Hidden Service.

Now, we shall edit the torrc file for the Hidden Service
vi /usr/local/etc/tor/torrc

and add the following lines

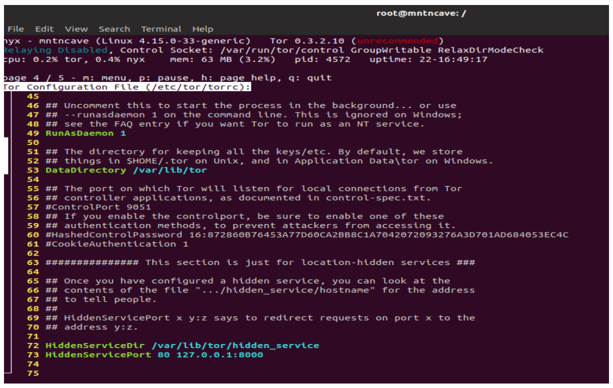
log info file/usr/local/etc/tor/tor.log
HiddenServiceDir /usr/local/etc/tor/hiddenservice/
HiddenServicePort 80 127.0.0.1:8000

This enables the server running on localhost port 8000 and make it available as a Tor Hidden Service.



```
nyx - mntncave (Linux 4.15.0-33-generic)   Tor 0.3.2.10 (unrecommended)
Relaying Disabled, Control Socket: /var/run/tor/control GroupWritable RelaxDirModeCheck
cpu: 0.0% tor, 0.4% nyx    mem: 63 MB (3.2%)   pid: 4572   uptime: 22-16:48:31

page 4 / 5 - m: menu, p: pause, h: page help, q: quit
Tor Configuration File (/etc/tor/torrc):

## Tor opens a SOCKS proxy on port 9050 by default -- even if you don't
## configure one below. Set "SOCKSPort 0" if you plan to run Tor only
## as a relay, and not make any local application connections yourself.
SOCKSPort 9050 # Default: Bind to localhost:9050 for local connections.
#SOCKSPort 192.168.0.1:9100 # Bind to this address:port too.

## Entry policies to allow/deny SOCKS requests based on IP address.
## First entry that matches wins. If no SOCKSPolicy is set, we accept
## all (and only) requests that reach a SOCKSPort. Untrusted users who
## can access your SOCKSPort may be able to learn about the connections
## you make.
#SOCKSPolicy accept 192.168.0.0/16
#SOCKSPolicy accept6 FC00::/7
#SOCKSPolicy reject *

## Logs go to stdout at level "notice" unless redirected by something
## else, like one of the below lines. You can have as many Log lines as
## you want.
##
## We advise using "notice" in most cases, since anything more verbose
## may provide sensitive information to an attacker who obtains the logs.
##
## Send all messages of level 'notice' or higher to /var/log/tor/notices.log
Log notice file /var/log/tor/notices.log
## Send every possible message to /var/log/tor/debug.log
Log debug file /var/log/tor/debug.log
## Use the system log instead of Tor's logfiles
#Log notice syslog
## To send all messages to stderr:
#Log debug stderr
```

Figure 4: Log configurations in torrc reflected in Nyx



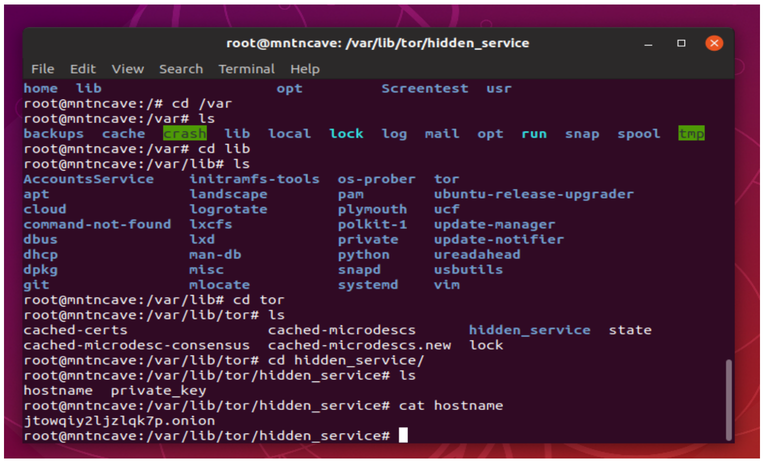
```
nyx - mntncave (Linux 4.15.0-33-generic)   Tor 0.3.2.10 (unrecommended)
Relaying Disabled, Control Socket: /var/run/tor/control GroupWritable RelaxDirModeCheck
cpu: 0.2% tor, 0.4% nyx    mem: 63 MB (3.2%)   pid: 4572   uptime: 22-16:49:17

page 4 / 5 - m: menu, p: pause, h: page help, q: quit
Tor Configuration File (/etc/tor/torrc):

## Uncomment this to start the process in the background... or use
## --runasdaemon 1 on the command line. This is ignored on Windows;
## see the FAQ entry if you want Tor to run as an NT service.
RunAsDaemon 1

## The directory for keeping all the keys/etc. By default, we store
## things in $HOME/.tor on Unix, and in Application Data\tor on Windows.
DataDirectory /var/lib/tor

## The port on which Tor will listen for local connections from Tor
## controller applications, as documented in control-spec.txt.
#ControlPort 9051
## If you enable the controlport, be sure to enable one of these
## authentication methods, to prevent attackers from accessing it.
#HashedControlPassword 16:872860B76453A77D60CA2BB8C1A7042072093276A3D701AD684053EC4C
#CookieAuthentication 1

############### This section is just for location-hidden services ###

## Once you have configured a hidden service, you can look at the
## contents of the file ".../hidden_service/hostname" for the address
## to tell people.
##
## HiddenServicePort x y:z says to redirect requests on port x to the
## address y:z.

HiddenServiceDir /var/lib/tor/hidden_service
HiddenServicePort 80 127.0.0.1:8000


```

Figure 5: Hidden Service configurations in torrc reflected in Nyx

Restart the Tor Service.

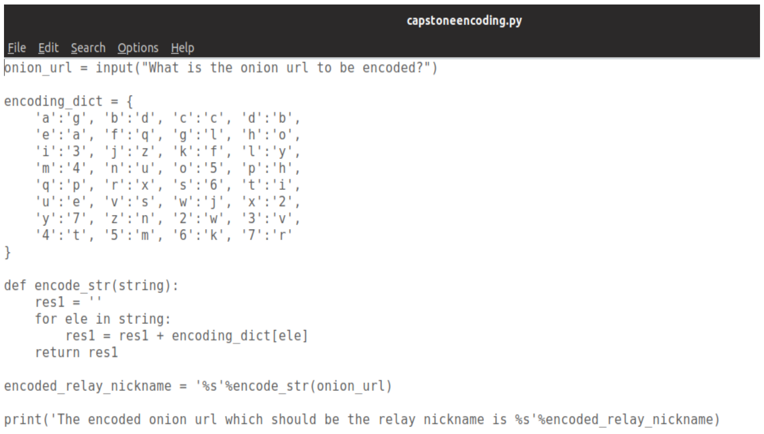
tor service restart

The next step is taking the onion URL and encoding it using the python code developed for the same.


root@mntncave: /var/lib/tor/hidden_service
File Edit View Search Terminal Help

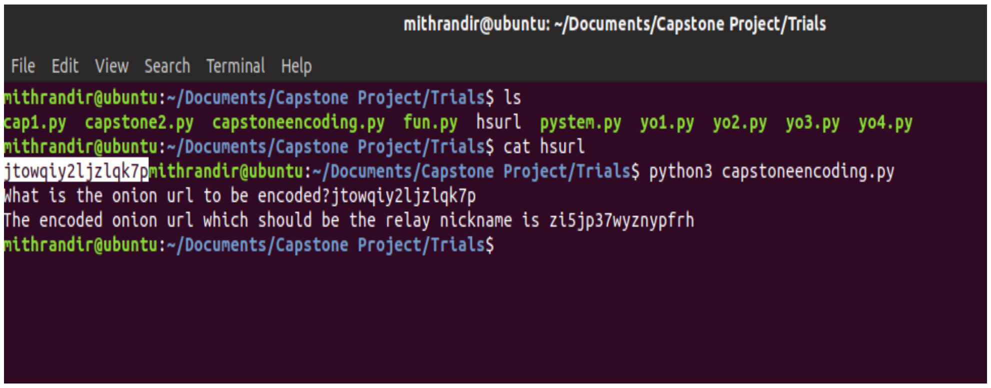
```
home  lib                    opt         Screentest  usr
root@mntncave:/# cd /var
root@mntncave:/var# ls
backups  cache  crash  lib  local  lock  log  mail  opt  run  snap  spool  tmp
root@mntncave:/var# cd lib
root@mntncave:/var/lib# ls
AccountsService    initramfs-tools  os-prober  tor
apt                landscape        pam        ubuntu-release-upgrader
cloud              logrotate        plymouth   ucf
command-not-found  lxcfs            polkit-1   update-manager
dbus               lxd              private    update-notifier
dhcp               man-db           python     ureadahead
dpkg               misc             snapd      usbutils
git                mlocate          systemd    vim
root@mntncave:/var/lib# cd tor
root@mntncave:/var/lib/tor# ls
cached-certs                cached-microdescs      hidden_service  state
cached-microdesc-consensus  cached-microdescs.new  lock
root@mntncave:/var/lib/tor# cd hidden_service/
root@mntncave:/var/lib/tor/hidden_service# ls
hostname  private_key
root@mntncave:/var/lib/tor/hidden_service# cat hostname
jtowqiy2ljzlqk7p.onion
root@mntncave:/var/lib/tor/hidden_service#
```

Figure 6: Onion URL being generated.The Onion URL will be encoded and made part of Bob's Relay nickname



```
onion_url = input("What is the onion url to be encoded?")

encoding_dict = {
    'a':'g', 'b':'d', 'c':'c', 'd':'b',
    'e':'a', 'f':'q', 'g':'l', 'h':'o',
    'i':'3', 'j':'z', 'k':'f', 'l':'y',
    'm':'4', 'n':'u', 'o':'5', 'p':'h',
    'q':'p', 'r':'x', 's':'6', 't':'i',
    'u':'e', 'v':'s', 'w':'j', 'x':'2',
    'y':'7', 'z':'n', '2':'w', '3':'v',
    '4':'t', '5':'m', '6':'k', '7':'r'
}

def encode_str(string):
    res1 = ''
    for ele in string:
        res1 = res1 + encoding_dict[ele]
    return res1

encoded_relay_nickname = '%s'%encode_str(onion_url)

print('The encoded onion url which should be the relay nickname is %s'%encoded_relay_nickname)
```

Figure 7: Python code for encoding the Onion URL

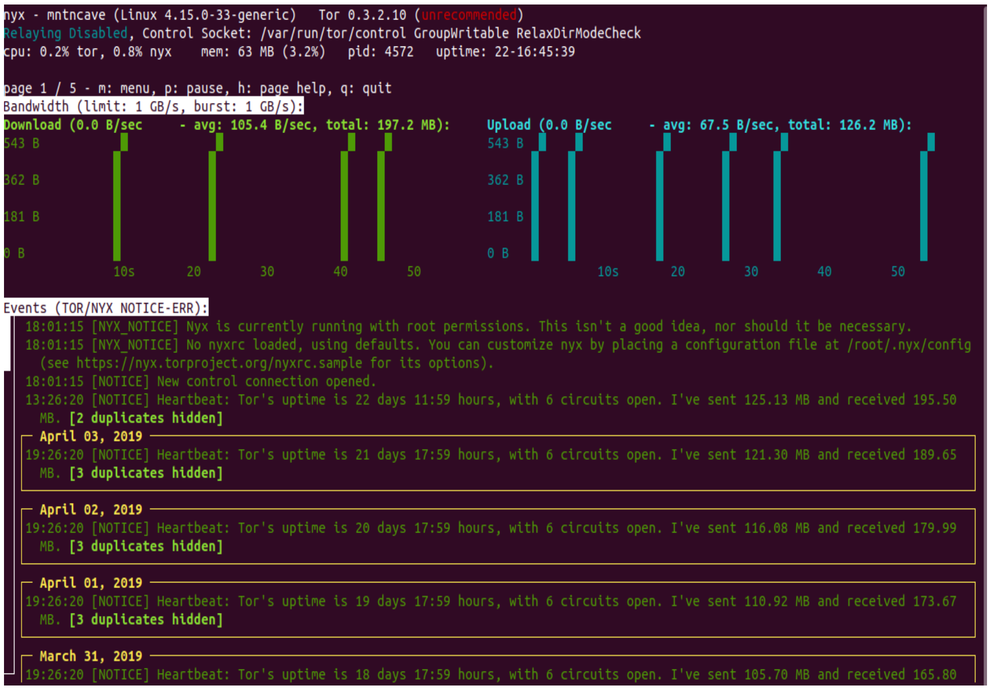


```
mithrandir@ubuntu: ~/Documents/Capstone Project/Trials
File  Edit  View  Search  Terminal  Help
mithrandir@ubuntu:~/Documents/Capstone Project/Trials$ ls
cap1.py  capstone2.py  capstoneencoding.py  fun.py  hsurl  pystem.py  yo1.py  yo2.py  yo3.py  yo4.py
mithrandir@ubuntu:~/Documents/Capstone Project/Trials$ cat hsurl
jtowqiy2ljzlqk7pmithrandir@ubuntu:~/Documents/Capstone Project/Trials$ python3 capstoneencoding.py
What is the onion url to be encoded?jtowqiy2ljzlqk7p
The encoded onion url which should be the relay nickname is zi5jp37wyznypfrh
mithrandir@ubuntu:~/Documents/Capstone Project/Trials$
```

Figure 8: Output of running the python code for encoding the Onion URL

The output of running the code is the string that should be the part of the torrc field Relay Nickname of Bob's Tor Relay.

Hence, the Hidden Service is finally functional



```
nyx - mntncave (Linux 4.15.0-33-generic)   Tor 0.3.2.10 (unrecommended)
Relaying Disabled, Control Socket: /var/run/tor/control GroupWritable RelaxDirModeCheck
cpu: 0.2% tor, 0.8% nyx    mem: 63 MB (3.2%)   pid: 4572   uptime: 22-16:45:39

page 1 / 5 - m: menu, p: pause, h: page help, q: quit
Bandwidth (limit: 1 GB/s, burst: 1 GB/s):
Download (0.0 B/sec     - avg: 105.4 B/sec, total: 197.2 MB):    Upload (0.0 B/sec     - avg: 67.5 B/sec, total: 126.2 MB):
543 B                                                            543 B

362 B                                                            362 B

181 B                                                            181 B

0 B                                                              0 B
               10s      20       30       40       50                          10s      20       30       40       50

Events (TOR/NYX NOTICE-ERR):
  18:01:15 [NYX_NOTICE] Nyx is currently running with root permissions. This isn't a good idea, nor should it be necessary.
  18:01:15 [NYX_NOTICE] No nyxrc loaded, using defaults. You can customize nyx by placing a configuration file at /root/.nyx/config
    (see https://nyx.torproject.org/nyxrc.sample for its options).
  18:01:15 [NOTICE] New control connection opened.
  13:26:20 [NOTICE] Heartbeat: Tor's uptime is 22 days 11:59 hours, with 6 circuits open. I've sent 125.13 MB and received 195.50
    MB. [2 duplicates hidden]
  April 03, 2019
  19:26:20 [NOTICE] Heartbeat: Tor's uptime is 21 days 17:59 hours, with 6 circuits open. I've sent 121.30 MB and received 189.65
    MB. [3 duplicates hidden]

  April 02, 2019
  19:26:20 [NOTICE] Heartbeat: Tor's uptime is 20 days 17:59 hours, with 6 circuits open. I've sent 116.08 MB and received 179.99
    MB. [3 duplicates hidden]

  April 01, 2019
  19:26:20 [NOTICE] Heartbeat: Tor's uptime is 19 days 17:59 hours, with 6 circuits open. I've sent 110.92 MB and received 173.67
    MB. [3 duplicates hidden]

  March 31, 2019
  19:26:20 [NOTICE] Heartbeat: Tor's uptime is 18 days 17:59 hours, with 6 circuits open. I've sent 105.70 MB and received 165.80
```

Figure 9: Hidden Service activity reflected in Nyx. The green and the blue spikes show bandwidth activity bursts.

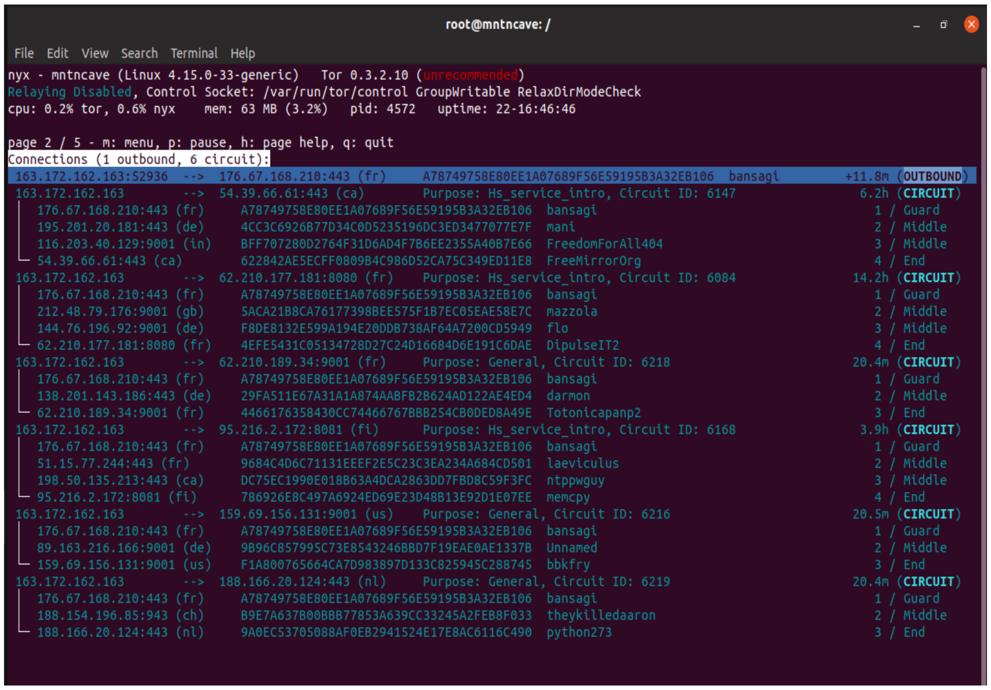

```
root@mntncave: /
File  Edit  View  Search  Terminal  Help
nyx - mntncave (Linux 4.15.0-33-generic)   Tor 0.3.2.10 (unrecommended)
Relaying Disabled, Control Socket: /var/run/tor/control GroupWritable RelaxDirModeCheck
cpu: 0.2% tor, 0.6% nyx    mem: 63 MB (3.2%)   pid: 4572   uptime: 22-16:46:46

page 2 / 5 - m: menu, p: pause, h: page help, q: quit
Connections (1 outbound, 6 circuit):
 163.172.162.163:52936  -->  176.67.168.210:443 (fr)     A78749758E80EE1A07689F56E59195B3A32EB106  bansagi         +11.8m (OUTBOUND)
 163.172.162.163        -->  54.39.66.61:443 (ca)        Purpose: Hs_service_intro, Circuit ID: 6147            6.2h (CIRCUIT)
 │  176.67.168.210:443 (fr)     A78749758E80EE1A07689F56E59195B3A32EB106  bansagi                             1 / Guard
 │  195.201.20.181:443 (de)     4CC3C6926B77D34C0D5235196DC3ED3477077E7F  mani                                2 / Middle
 │  116.203.40.129:9001 (in)    BFF707280D2764F31D6AD4F7B6EE2355A40B7E66  FreedomForAll404                    3 / Middle
 └─ 54.39.66.61:443 (ca)        622842AE5ECFF0809B4C986D52CA75C349ED11E8  FreeMirrorOrg                       4 / End
 163.172.162.163        -->  62.210.177.181:8080 (fr)    Purpose: Hs_service_intro, Circuit ID: 6084           14.2h (CIRCUIT)
 │  176.67.168.210:443 (fr)     A78749758E80EE1A07689F56E59195B3A32EB106  bansagi                             1 / Guard
 │  212.48.79.176:9001 (gb)     5ACA21B8CA76177398BEE575F1B7EC05EAE58E7C  mazzola                             2 / Middle
 │  144.76.196.92:9001 (de)     F8DE8132E599A194E20DDB738AF64A7200CD5949  flo                                 3 / Middle
 └─ 62.210.177.181:8080 (fr)    4EFE5431C05134728D27C24D16684D6E191C6DAE  DipulseIT2                          4 / End
 163.172.162.163        -->  62.210.189.34:9001 (fr)     Purpose: General, Circuit ID: 6218                    20.4m (CIRCUIT)
 │  176.67.168.210:443 (fr)     A78749758E80EE1A07689F56E59195B3A32EB106  bansagi                             1 / Guard
 │  138.201.143.186:443 (de)    29FA511E67A31A1A874AABFB2B624AD122AE4ED4  darmon                              2 / Middle
 └─ 62.210.189.34:9001 (fr)     4466176358430CC74466767BBB254CB0DED8A49E  Totonicapanp2                       3 / End
 163.172.162.163        -->  95.216.2.172:8081 (fi)      Purpose: Hs_service_intro, Circuit ID: 6168            3.9h (CIRCUIT)
 │  176.67.168.210:443 (fr)     A78749758E80EE1A07689F56E59195B3A32EB106  bansagi                             1 / Guard
 │  51.15.77.244:443 (fr)       9684C4D6C71131EEEF2E5C23C3EA234A684CD501  laeviculus                          2 / Middle
 │  198.50.135.213:443 (ca)     DC75EC1990E018B63A4DCA2863DD7FBD8C59F3FC  ntppwguy                            3 / Middle
 └─ 95.216.2.172:8081 (fi)      786926E8C497A6924ED69E23D48B13E92D1E07EE  memcpy                              4 / End
 163.172.162.163        -->  159.69.156.131:9001 (us)    Purpose: General, Circuit ID: 6216                    20.5m (CIRCUIT)
 │  176.67.168.210:443 (fr)     A78749758E80EE1A07689F56E59195B3A32EB106  bansagi                             1 / Guard
 │  89.163.216.166:9001 (de)    9B96C857995C73E8543246BBD7F19EAE0AE1337B  Unnamed                             2 / Middle
 └─ 159.69.156.131:9001 (us)    F1A800765664CA7D983897D133C825945C288745  bbkfry                              3 / End
 163.172.162.163        -->  188.166.20.124:443 (nl)     Purpose: General, Circuit ID: 6219                    20.4m (CIRCUIT)
 │  176.67.168.210:443 (fr)     A78749758E80EE1A07689F56E59195B3A32EB106  bansagi                             1 / Guard
 │  188.154.196.85:943 (ch)     B9E7A637B00BBB77853A639CC33245A2FEB8F033  theykilledaaron                     2 / Middle
 └─ 188.166.20.124:443 (nl)     9A0EC53705088AF0EB2941524E17E8AC6116C490  python273                           3 / End
```

Figure 10: Hidden Service establishing circuits in the Tor network. The blue fields can be expanded to reveal which Relays are used in establishing the Introduction and Rendezvous circuits by the Tor Relay

The second step in the implementation of the obscure communication system is setting up the Tor Relay in the following steps :

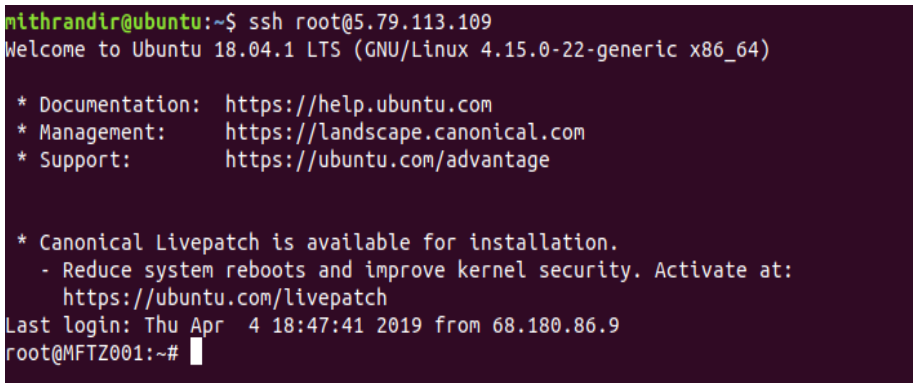

```
mithrandir@ubuntu:~$ ssh root@5.79.113.109
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-22-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage


 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch
Last login: Thu Apr  4 18:47:41 2019 from 68.180.86.9
root@MFTZ001:~# 
```

Figure 11: ssh connection to the Leaseweb cloud service provider where the Tor Relay is hosted

Installing Tor as the first step : Install Tor using the following commands

```
sudo apt-get install tor
service tor start
```

It is important that the Tor Relay keeps accurate time, so we will change the timezone and setup the ntp client. First, listing the timezones shall help us find the correct one corresponding to the location of the machine

```
timedatectl list-timezones
```

Set the timezone to the one for the machine's location. Amsterdam is used as an example here.

```
sudo timedatectl set-timezone Europe/Amsterdam
```

Install ntp

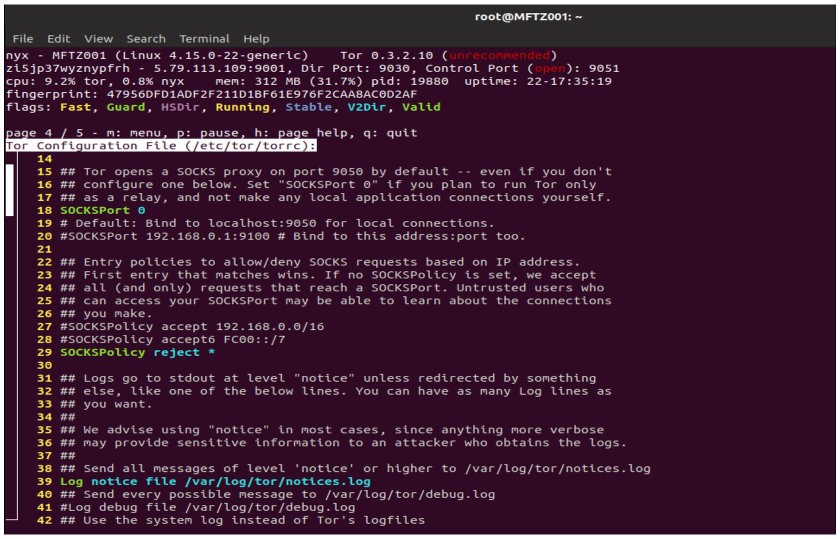
```
sudo apt-get install ntp
```

Now we shall configure the Relay by editing the torrc file

```
sudo nano /etc/tor/torrc
```

(Continued on the next page)

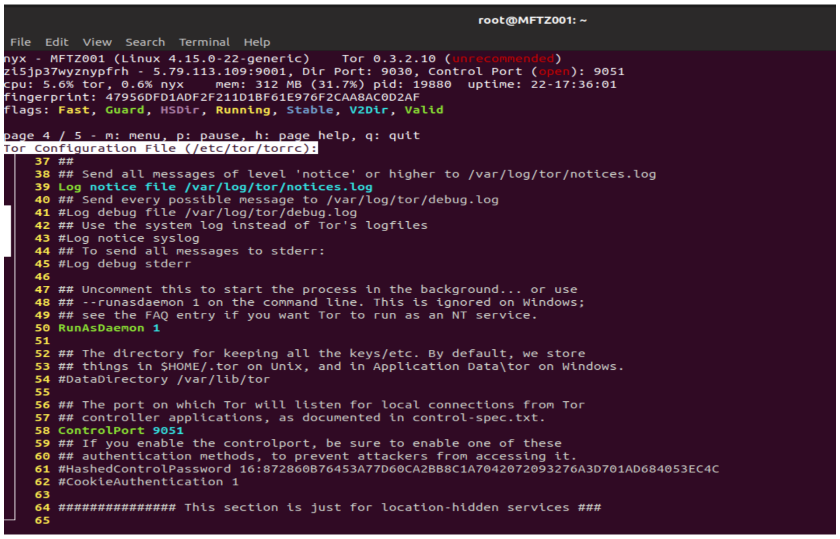
We do not need a SOCKS proxy, so we shall uncomment the line
SOCKSPolicy reject *



```
nyx - MFTZ001 (Linux 4.15.0-22-generic)    Tor 0.3.2.10 (unrecommended)
zi5jp37wyznypfrh - 5.79.113.109:9001, Dir Port: 9030, Control Port (open): 9051
cpu: 9.2% tor, 0.8% nyx    mem: 312 MB (31.7%) pid: 19880  uptime: 22-17:35:19
fingerprint: 47956DFD1ADF2F211D1BF61E976F2CAA8AC0D2AF
flags: Fast, Guard, HSDir, Running, Stable, V2Dir, Valid

page 4 / 5 - m: menu, p: pause, h: page help, q: quit
Tor Configuration File (/etc/tor/torrc):

## Tor opens a SOCKS proxy on port 9050 by default -- even if you don't
## configure one below. Set "SOCKSPort 0" if you plan to run Tor only
## as a relay, and not make any local application connections yourself.
SOCKSPort 0
# Default: Bind to localhost:9050 for local connections.
#SOCKSPort 192.168.0.1:9100 # Bind to this address:port too.

## Entry policies to allow/deny SOCKS requests based on IP address.
## First entry that matches wins. If no SOCKSPolicy is set, we accept
## all (and only) requests that reach a SOCKSPort. Untrusted users who
## can access your SOCKSPort may be able to learn about the connections
## you make.
#SOCKSPolicy accept 192.168.0.0/16
#SOCKSPolicy accept6 FC00::/7
SOCKSPolicy reject *

## Logs go to stdout at level "notice" unless redirected by something
## else, like one of the below lines. You can have as many Log lines as
## you want.
##
## We advise using "notice" in most cases, since anything more verbose
## may provide sensitive information to an attacker who obtains the logs.
##
## Send all messages of level 'notice' or higher to /var/log/tor/notices.log
Log notice file /var/log/tor/notices.log
## Send every possible message to /var/log/tor/debug.log
#Log debug file /var/log/tor/debug.log
## Use the system log instead of Tor's logfiles
```

Figure 12: Socks Policy and Socks Port reflected in torrc

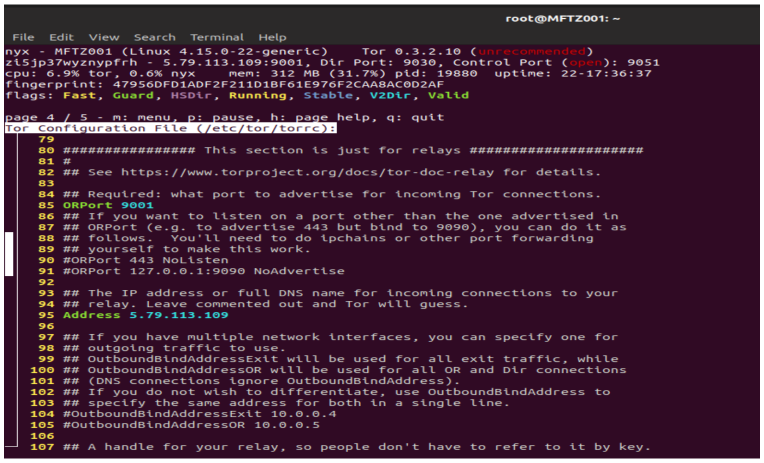
We will be running as a daemon, so uncomment the line
RunAsDaemon 1



```
nyx - MFTZ001 (Linux 4.15.0-22-generic)    Tor 0.3.2.10 (unrecommended)
zi5jp37wyznypfrh - 5.79.113.109:9001, Dir Port: 9030, Control Port (open): 9051
cpu: 5.6% tor, 0.6% nyx    mem: 312 MB (31.7%) pid: 19880  uptime: 22-17:36:01
fingerprint: 47956DFD1ADF2F211D1BF61E976F2CAA8AC0D2AF
flags: Fast, Guard, HSDir, Running, Stable, V2Dir, Valid

page 4 / 5 - m: menu, p: pause, h: page help, q: quit
Tor Configuration File (/etc/tor/torrc):
##
## Send all messages of level 'notice' or higher to /var/log/tor/notices.log
Log notice file /var/log/tor/notices.log
## Send every possible message to /var/log/tor/debug.log
#Log debug file /var/log/tor/debug.log
## Use the system log instead of Tor's logfiles
#Log notice syslog
## To send all messages to stderr:
#Log debug stderr

## Uncomment this to start the process in the background... or use
## --runasdaemon 1 on the command line. This is ignored on Windows;
## see the FAQ entry if you want Tor to run as an NT service.
RunAsDaemon 1

## The directory for keeping all the keys/etc. By default, we store
## things in $HOME/.tor on Unix, and in Application Data\tor on Windows.
#DataDirectory /var/lib/tor

## The port on which Tor will listen for local connections from Tor
## controller applications, as documented in control-spec.txt.
ControlPort 9051
## If you enable the controlport, be sure to enable one of these
## authentication methods, to prevent attackers from accessing it.
#HashedControlPassword 16:872860B76453A77D60CA2BB8C1A7042072093276A3D701AD684053EC4C
#CookieAuthentication 1

############### This section is just for location-hidden services ###

```

Figure 13: RunAsDaemon, ControlPort and Logging Options in torrc

Relay needs an ORPort for incoming connections, so uncomment the line
ORPort 9001

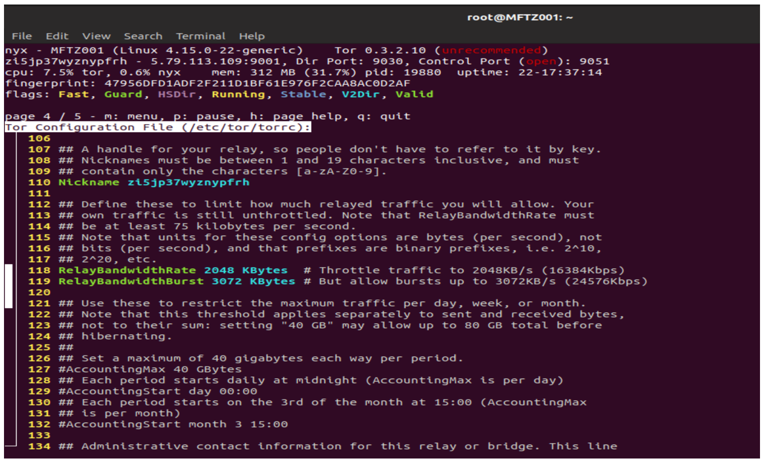


```
nyx - MFTZ001 (Linux 4.15.0-22-generic)    Tor 0.3.2.10 (unrecommended)
zi5jp37wyznypfrh - 5.79.113.109:9001, Dir Port: 9030, Control Port (open): 9051
cpu: 6.9% tor, 0.6% nyx    mem: 312 MB (31.7%) pid: 19880  uptime: 22-17:36:37
fingerprint: 47956DFD1ADF2F211D1BF61E976F2CAA8AC0D2AF
flags: Fast, Guard, HSDir, Running, Stable, V2Dir, Valid

page 4 / 5 - m: menu, p: pause, h: page help, q: quit
Tor Configuration File (/etc/tor/torrc):

################ This section is just for relays #####################
#
## See https://www.torproject.org/docs/tor-doc-relay for details.

## Required: what port to advertise for incoming Tor connections.
ORPort 9001
## If you want to listen on a port other than the one advertised in
## ORPort (e.g. to advertise 443 but bind to 9090), you can do it as
## follows.  You'll need to do ipchains or other port forwarding
## yourself to make this work.
#ORPort 443 NoListen
#ORPort 127.0.0.1:9090 NoAdvertise

## The IP address or full DNS name for incoming connections to your
## relay. Leave commented out and Tor will guess.
Address 5.79.113.109

## If you have multiple network interfaces, you can specify one for
## outgoing traffic to use.
## OutboundBindAddressExit will be used for all exit traffic, while
## OutboundBindAddressOR will be used for all OR and Dir connections
## (DNS connections ignore OutboundBindAddress).
## If you do not wish to differentiate, use OutboundBindAddress to
## specify the same address for both in a single line.
#OutboundBindAddressExit 10.0.0.4
#OutboundBindAddressOR 10.0.0.5

## A handle for your relay, so people don't have to refer to it by key.
```

Figure 14: ORPort and Relay Address in torrc

The Relay should have a nickname as this step is critical for Alice. Alice shall be using the Python based Stem-utility for fetching the Relay nickname and decode it to obtain the obscured onion URL

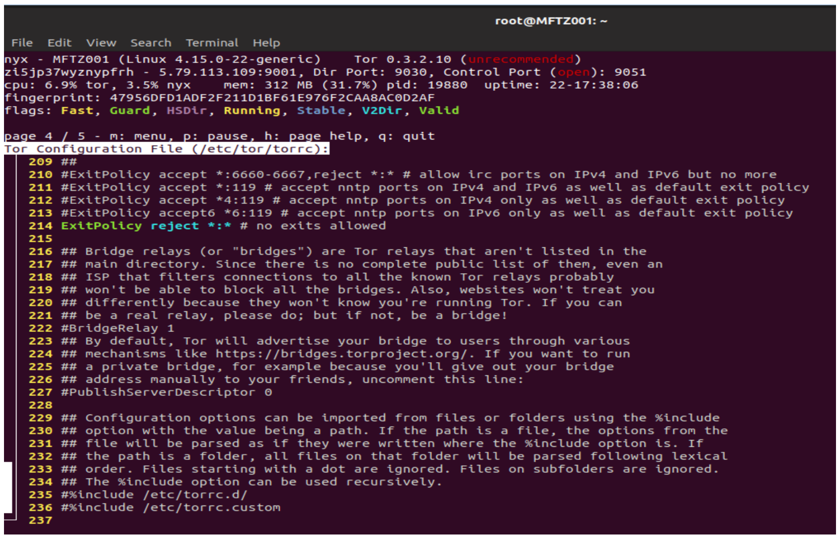
Nickname zi5jp37wyznypfrh



```
nyx - MFTZ001 (Linux 4.15.0-22-generic)    Tor 0.3.2.10 (unrecommended)
zi5jp37wyznypfrh - 5.79.113.109:9001, Dir Port: 9030, Control Port (open): 9051
cpu: 7.5% tor, 0.6% nyx    mem: 312 MB (31.7%) pid: 19880  uptime: 22-17:37:14
fingerprint: 47956DFD1ADF2F211D1BF61E976F2CAA8AC0D2AF
flags: Fast, Guard, HSDir, Running, Stable, V2Dir, Valid

page 4 / 5 - m: menu, p: pause, h: page help, q: quit
Tor Configuration File (/etc/tor/torrc):

## A handle for your relay, so people don't have to refer to it by key.
## Nicknames must be between 1 and 19 characters inclusive, and must
## contain only the characters [a-zA-Z0-9].
Nickname zi5jp37wyznypfrh

## Define these to limit how much relayed traffic you will allow. Your
## own traffic is still unthrottled. Note that RelayBandwidthRate must
## be at least 75 kilobytes per second.
## Note that units for these config options are bytes (per second), not
## bits (per second), and that prefixes are binary prefixes, i.e. 2^10,
## 2^20, etc.
RelayBandwidthRate 2048 KBytes  # Throttle traffic to 2048KB/s (16384Kbps)
RelayBandwidthBurst 3072 KBytes # But allow bursts up to 3072KB/s (24576Kbps)

## Use these to restrict the maximum traffic per day, week, or month.
## Note that this threshold applies separately to sent and received bytes,
## not to their sum: setting "40 GB" may allow up to 80 GB total before
## hibernating.
##
## Set a maximum of 40 gigabytes each way per period.
#AccountingMax 40 GBytes
## Each period starts daily at midnight (AccountingMax is per day)
#AccountingStart day 00:00
## Each period starts on the 3rd of the month at 15:00 (AccountingMax
## is per month)
#AccountingStart month 3 15:00

## Administrative contact information for this relay or bridge. This line
```

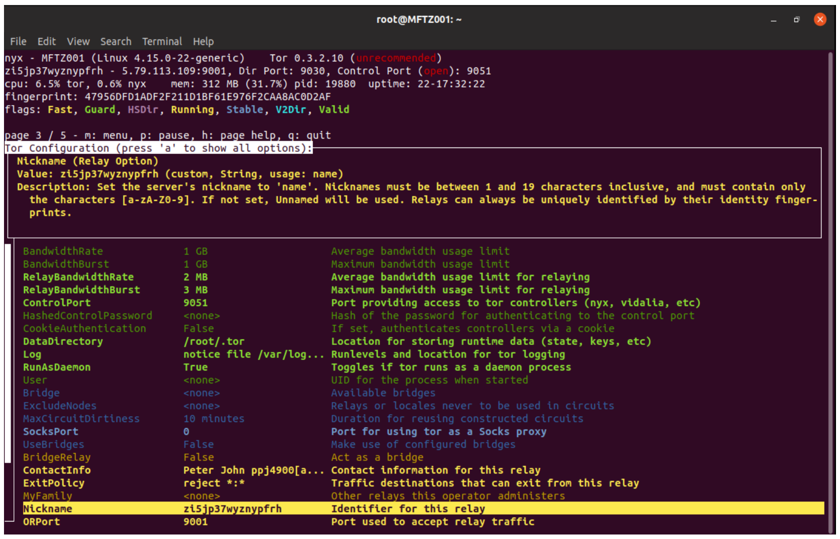
Figure 15: Nickname and Relay Bandwidth in torrc

The other most important option is for disallowing exits, hence we should uncomment the line
ExitPolicy reject *:*



```
nyx - MFTZ001 (Linux 4.15.0-22-generic)    Tor 0.3.2.10 (unrecommended)
zi5jp37wyznypfrh - 5.79.113.109:9001, Dir Port: 9030, Control Port (open): 9051
cpu: 6.9% tor, 3.5% nyx    mem: 312 MB (31.7%) pid: 19880  uptime: 22-17:38:06
fingerprint: 47956DFD1ADF2F211D1BF61E976F2CAA8AC0D2AF
flags: Fast, Guard, HSDir, Running, Stable, V2Dir, Valid

page 4 / 5 - m: menu, p: pause, h: page help, q: quit
Tor Configuration File (/etc/tor/torrc):
  ##
  #ExitPolicy accept *:6660-6667,reject *:* # allow irc ports on IPv4 and IPv6 but no more
  #ExitPolicy accept *:119 # accept nntp ports on IPv4 and IPv6 as well as default exit policy
  #ExitPolicy accept *4:119 # accept nntp ports on IPv4 only as well as default exit policy
  #ExitPolicy accept6 *6:119 # accept nntp ports on IPv6 only as well as default exit policy
  ExitPolicy reject *:* # no exits allowed

  ## Bridge relays (or "bridges") are Tor relays that aren't listed in the
  ## main directory. Since there is no complete public list of them, even an
  ## ISP that filters connections to all the known Tor relays probably
  ## won't be able to block all the bridges. Also, websites won't treat you
  ## differently because they won't know you're running Tor. If you can
  ## be a real relay, please do; but if not, be a bridge!
  #BridgeRelay 1
  ## By default, Tor will advertise your bridge to users through various
  ## mechanisms like https://bridges.torproject.org/. If you want to run
  ## a private bridge, for example because you'll give out your bridge
  ## address manually to your friends, uncomment this line:
  #PublishServerDescriptor 0

  ## Configuration options can be imported from files or folders using the %include
  ## option with the value being a path. If the path is a file, the options from the
  ## file will be parsed as if they were written where the %include option is. If
  ## the path is a folder, all files on that folder will be parsed following lexical
  ## order. Files starting with a dot are ignored. Files on subfolders are ignored.
  ## The %include option can be used recursively.
  #%include /etc/torrc.d/
  #%include /etc/torrc.custom

```

Figure 16: Exit Policy changes reflected in torrc



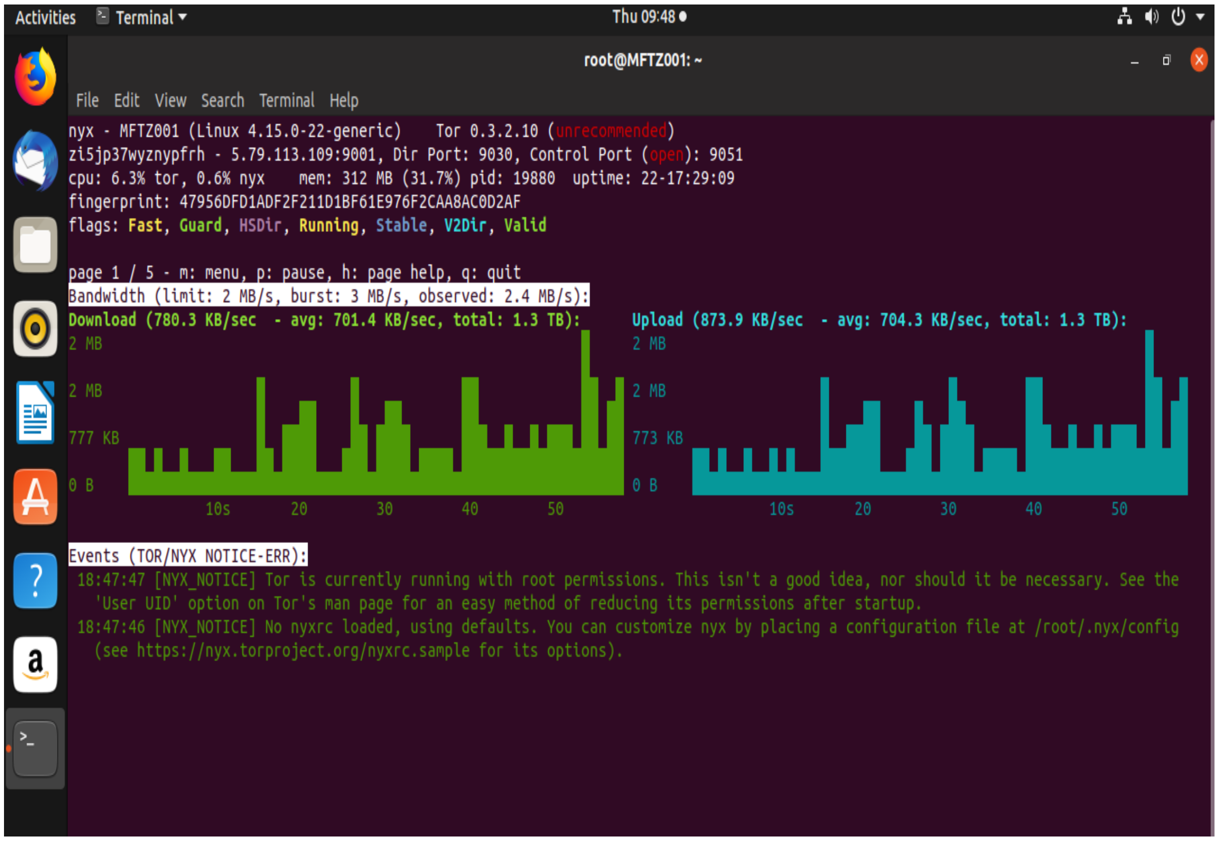
```
nyx - MFTZ001 (Linux 4.15.0-22-generic)    Tor 0.3.2.10 (unrecommended)
zi5jp37wyznypfrh - 5.79.113.109:9001, Dir Port: 9030, Control Port (open): 9051
cpu: 6.5% tor, 0.6% nyx    mem: 312 MB (31.7%) pid: 19880  uptime: 22-17:32:22
fingerprint: 47956DFD1ADF2F211D1BF61E976F2CAA8AC0D2AF
flags: Fast, Guard, HSDir, Running, Stable, V2Dir, Valid

page 3 / 5 - m: menu, p: pause, h: page help, q: quit
Tor Configuration (press 'a' to show all options):
 Nickname (Relay Option)
 Value: zi5jp37wyznypfrh (custom, String, usage: name)
 Description: Set the server's nickname to 'name'. Nicknames must be between 1 and 19 characters inclusive, and must contain only
   the characters [a-zA-Z0-9]. If not set, Unnamed will be used. Relays can always be uniquely identified by their identity finger-
   prints.

  BandwidthRate            1 GB                     Average bandwidth usage limit
  BandwidthBurst           1 GB                     Maximum bandwidth usage limit
  RelayBandwidthRate       2 MB                     Average bandwidth usage limit for relaying
  RelayBandwidthBurst      3 MB                     Maximum bandwidth usage limit for relaying
  ControlPort              9051                     Port providing access to tor controllers (nyx, vidalia, etc)
  HashedControlPassword    <none>                   Hash of the password for authenticating to the control port
  CookieAuthentication     False                    If set, authenticates controllers via a cookie
  DataDirectory            /root/.tor               Location for storing runtime data (state, keys, etc)
  Log                      notice file /var/log...  Runlevels and location for tor logging
  RunAsDaemon              True                     Toggles if tor runs as a daemon process
  User                     <none>                   UID for the process when started
  Bridge                   <none>                   Available bridges
  ExcludeNodes             <none>                   Relays or locales never to be used in circuits
  MaxCircuitDirtiness      10 minutes               Duration for reusing constructed circuits
  SocksPort                0                        Port for using tor as a Socks proxy
  UseBridges               False                    Make use of configured bridges
  BridgeRelay              False                    Act as a bridge
  ContactInfo              Peter John ppj4900[a...  Contact information for this relay
  ExitPolicy               reject *:*               Traffic destinations that can exit from this relay
  MyFamily                 <none>                   Other relays this operator administers
  Nickname                 zi5jp37wyznypfrh         Identifier for this relay
  ORPort                   9001                     Port used to accept relay traffic
```

Figure 17: Relay nickname a part of consensus and torrc

After saving the file, run the following command
sudo service tor restart

We shall be monitoring the Relay using the "Nyx" utility which we shall install using the following commands :

sudo apt-get install python-setuptools
sudo easy_install pip
sudo pip install nyx



```
nyx - MFTZ001 (Linux 4.15.0-22-generic)    Tor 0.3.2.10 (unrecommended)
zi5jp37wyznypfrh - 5.79.113.109:9001, Dir Port: 9030, Control Port (open): 9051
cpu: 6.3% tor, 0.6% nyx    mem: 312 MB (31.7%) pid: 19880  uptime: 22-17:29:09
fingerprint: 47956DFD1ADF2F211D1BF61E976F2CAA8AC0D2AF
flags: Fast, Guard, HSDir, Running, Stable, V2Dir, Valid

page 1 / 5 - m: menu, p: pause, h: page help, q: quit
Bandwidth (limit: 2 MB/s, burst: 3 MB/s, observed: 2.4 MB/s):
Download (780.3 KB/sec  - avg: 701.4 KB/sec, total: 1.3 TB):     Upload (873.9 KB/sec  - avg: 704.3 KB/sec, total: 1.3 TB):
2 MB                                                             2 MB

2 MB                                                             2 MB

777 KB                                                           773 KB

0 B                                                              0 B
          10s     20      30      40      50                               10s     20      30      40      50

Events (TOR/NYX NOTICE-ERR):
 18:47:47 [NYX_NOTICE] Tor is currently running with root permissions. This isn't a good idea, nor should it be necessary. See the
   'User UID' option on Tor's man page for an easy method of reducing its permissions after startup.
 18:47:46 [NYX_NOTICE] No nyxrc loaded, using defaults. You can customize nyx by placing a configuration file at /root/.nyx/config
   (see https://nyx.torproject.org/nyxrc.sample for its options).
```

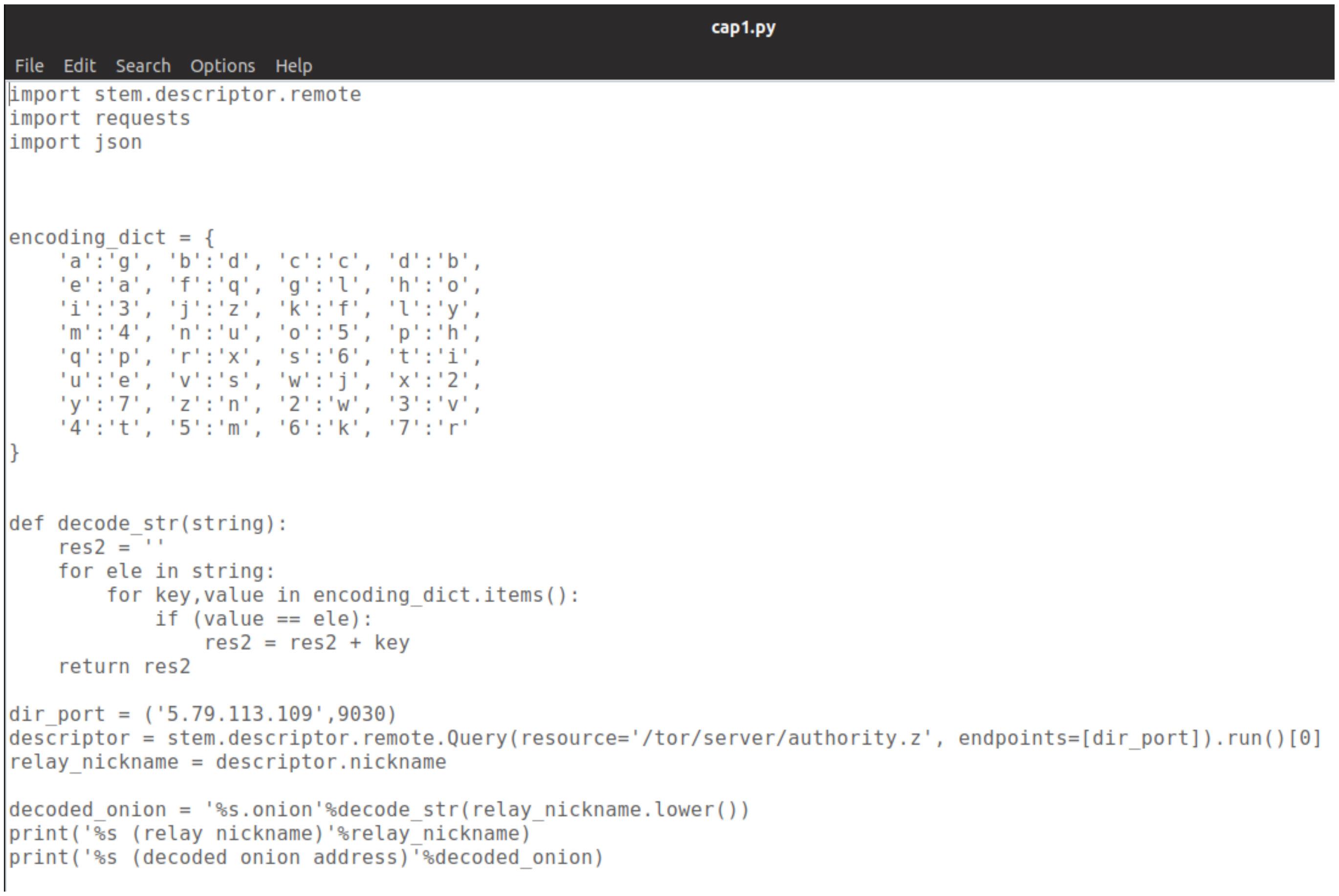
Figure 18: Nyx output for Relay Activity. It shows the bandwidth and various characteristics of the Tor Relay like Nickname, Fingerprint, Flags etc.

6 Testing and Experiments

Now that we have setup the Hidden Service and the Tor Relay, we can run the python script developed to test the communication system and obtain the Relay Nickname, decode it to get the obscured Onion URL which can be accessed from the shell or using the Tor Browser.

The code is run at Alice's end and deploys the STEM utility developed in Python which is an implementation of Tor Control specification. Alice obtains the Relay nickname and decodes it obtain the Onion URL of Bob's Hidden Service. Alice then accesses the Hidden Service using a Tor Browser.

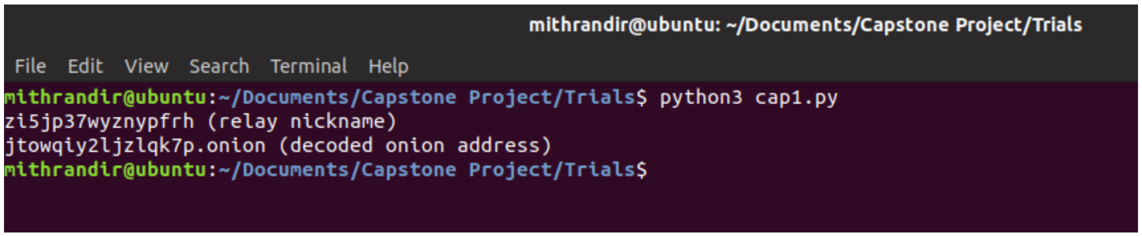
The following is the code used



```
import stem.descriptor.remote
import requests
import json



encoding_dict = {
    'a':'g', 'b':'d', 'c':'c', 'd':'b',
    'e':'a', 'f':'q', 'g':'l', 'h':'o',
    'i':'3', 'j':'z', 'k':'f', 'l':'y',
    'm':'4', 'n':'u', 'o':'5', 'p':'h',
    'q':'p', 'r':'x', 's':'6', 't':'i',
    'u':'e', 'v':'s', 'w':'j', 'x':'2',
    'y':'7', 'z':'n', '2':'w', '3':'v',
    '4':'t', '5':'m', '6':'k', '7':'r'
}


def decode_str(string):
    res2 = ''
    for ele in string:
        for key,value in encoding_dict.items():
            if (value == ele):
                res2 = res2 + key
    return res2

dir_port = ('5.79.113.109',9030)
descriptor = stem.descriptor.remote.Query(resource='/tor/server/authority.z', endpoints=[dir_port]).run()[0]
relay_nickname = descriptor.nickname

decoded_onion = '%s.onion'%decode_str(relay_nickname.lower())
print('%s (relay nickname)'%relay_nickname)
print('%s (decoded onion address)'%decoded_onion)
```

Figure 19: Python code for testing the obscure communication system

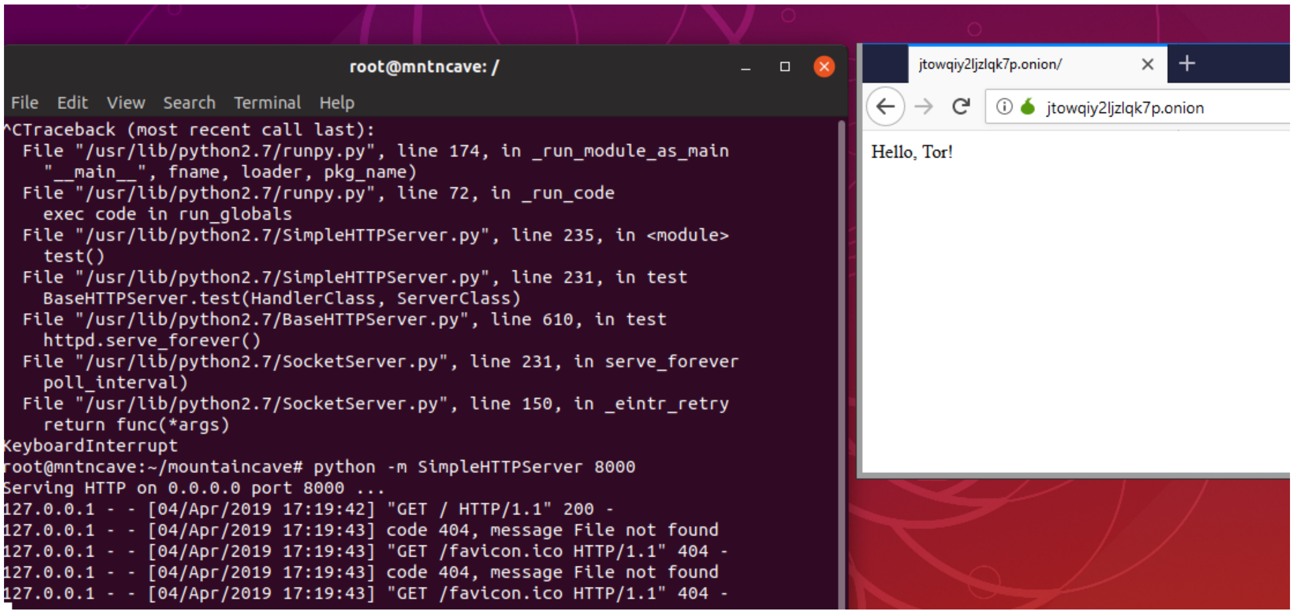
The following is the testing result of running the code. It signifies that Alice obtained the Relay nickname by accessing the metadata for Bob's Relay from the Tor consensus in the Tor network and decoded it to get the onion URL for Bob's Hidden Service hence accomplishing the goal of this capstone project for obscuring the process of Onion URL distribution out-of-band and accessing the Hidden Service.


mithrandir@ubuntu: ~/Documents/Capstone Project/Trials
File Edit View Search Terminal Help

```
mithrandir@ubuntu:~/Documents/Capstone Project/Trials$ python3 cap1.py
zi5jp37wyznypfrh (relay nickname)
jtowqiy2ljzlqk7p.onion (decoded onion address)
mithrandir@ubuntu:~/Documents/Capstone Project/Trials$
```

Figure 20: Test result

The following is the final step of accessing the Hidden Service using the onion URL obtained in the above step



```
^CTraceback (most recent call last):
  File "/usr/lib/python2.7/runpy.py", line 174, in _run_module_as_main
    "__main__", fname, loader, pkg_name)
  File "/usr/lib/python2.7/runpy.py", line 72, in _run_code
    exec code in run_globals
  File "/usr/lib/python2.7/SimpleHTTPServer.py", line 235, in <module>
    test()
  File "/usr/lib/python2.7/SimpleHTTPServer.py", line 231, in test
    BaseHTTPServer.test(HandlerClass, ServerClass)
  File "/usr/lib/python2.7/BaseHTTPServer.py", line 610, in test
    httpd.serve_forever()
  File "/usr/lib/python2.7/SocketServer.py", line 231, in serve_forever
    poll_interval)
  File "/usr/lib/python2.7/SocketServer.py", line 150, in _eintr_retry
    return func(*args)
KeyboardInterrupt
root@mntncave:~/mountaincave# python -m SimpleHTTPServer 8000
Serving HTTP on 0.0.0.0 port 8000 ...
127.0.0.1 - - [04/Apr/2019 17:19:42] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Apr/2019 17:19:43] code 404, message File not found
127.0.0.1 - - [04/Apr/2019 17:19:43] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [04/Apr/2019 17:19:43] code 404, message File not found
127.0.0.1 - - [04/Apr/2019 17:19:43] "GET /favicon.ico HTTP/1.1" 404 -
```

Hello, Tor!

Figure 21: Accessing the Hidden service from the above result and logs generated at the Hidden Service

7 Threat Model

Biryukov et. al.[7] studied the security of descriptor distribution procedure for Tor Hidden Services and showed that an adversary can attain complete control over the distribution of a particular Hidden Service.

The researchers used the following process for attacking the hidden service descriptor distribution process. The attacker would compute the descriptor IDs of the hidden service for any future moment and compute the fingerprints of expected responsible HS directories.

After that the attacker would compute the private/public key pairs so that the SHA-1 hash of the public-keys would be in-between the descriptor ID and the fingerprint of the first responsible hidden service directory. The attacker would then run Tor Relays with the precomputed public/private key pair and wait for 25 hours till the relays obtain the HSDir flag.

As the attackers relays appear in the consensus, the hidden service would use them and upload the descriptors which will then be downloaded by the clients to access the hidden service.

In this manner, by injecting 6 Tor Relays with precomputed public keys, the attacker gains control over all the responsible HSDirs for a particular Hidden Service.

8 Conclusion

This capstone explores a communication system for accessing a Hidden Service which adds a layer of obscurity to the communication process between the entity responsible for hosting the hidden service(Bob) and the entity accessing the Hidden Service(Alice). Instead of out-of-band Onion URL distribution, the onion URL is distributed in an obscured fashion not making it available to the public.

Complete control over the communication system can be made possible if Bob and Alice also control the Hidden Service directories which was explored by Biryukov et. al.[7]

A Tor Relay and a Tor Hidden Service were hosted on different cloud service providers to implement this project. It was an enriching experience to learn the nuances of hosting a live Tor Relay which was a part of the Tor consensus. Code was deployed in Python for encoding/decoding and the STEM utility, which is a Python implementation of the Tor Control specification, was used at Alice's end to complete the communication chain for accessing Bob's Hidden Service.

Overall, the project was inspired by the use of Tor metadata and Tor infrastructure to implement obscure/covert communication.

9 Acknowledgment

I thank my advisor Dr.Sumita Mishra for the guidance and I appreciate the financial support from my family.

I would like to thank all my professors and friends who have been with me throughout my Masters in Computing Security learning experience at Rochester Institute of Technology.

I would like to dedicate this project to Goddess Mother Tripura Sundari Ma who always inspires and blesses me in my undertakings.

References

[1] B. Lampson, “A note on the confinement problem,” 1973.

[2] J. Gimbi, D. Johnson, P. Lutz, and B. Yuan, “A covert channel over transport layer source ports,” 2012.

[3] A. Mileva and B. Panajotov, “Covert channels in tcp/ip protocol stack-extended version,” *Open Computer Science*, vol. 4, no. 2, pp. 45–66, 2014.

[4] D. Anthony, D. Johnson, P. Lutz, and B. Yuan, “A behavior based covert channel within anti-virus updates,” 2012.

[5] K. Stokes, B. Yuan, D. Johnson, and P. Lutz, “Icmp covert channel resiliency,” in *Technological Developments in Networking, Education and Automation.* Springer, 2010, pp. 503–506.

[6] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas, “Circuit fingerprinting attacks: Passive deanonymization of tor hidden services,” in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 287–302.

[7] A. Biryukov, I. Pustogarov, and R.-P. Weinmann, “Trawling for tor hidden services: Detection, measurement, deanonymization,” in *2013 IEEE Symposium on Security and Privacy.* IEEE, 2013, pp. 80–94.

[8] A. Biryukov, I. Pustogarov, F. Thill, and R.-P. Weinmann, “Content and popularity analysis of tor hidden services,” in *2014 IEEE 34th International Conference on Distributed Computing Systems Workshops (ICDCSW).* IEEE, 2014, pp. 188–193.

[9] A. Iacovazzi, S. Sarda, D. Frassinelli, and Y. Elovici, “Dropwat: an invisible network flow watermark for data exfiltration traceback,” *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1139–1154, 2018.

[10] A. Iacovazzi, S. Sarda, and Y. Elovici, “Inflow: Inverse network flow watermarking for detecting hidden servers,” in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications.* IEEE, 2018, pp. 747–755.