MeowOnion Proxy

COMPSCI 512 Distributed Systems - Fall 2024

Marcus Ortiz, Alexis Cruz-Ayala, Rasti Aldawoodi

November 25th, 2024

I. Introduction

As the rapid expansion of internet connected devices continues to automate and improve various tasks, it also infringes on our daily lives. Therefore, it is vital that anonymous internet connections are offered to protect our right to privacy. Although the shielding of illegal and immoral activity is a known and valid argument against anonymization networks, these networks also encourage activism, whistleblowing, research, and journalism without fear of repercussion from oppressive forces. Therefore, this project is designing and implementing a simplified version of Tor/Onion Routing, MeowOnion Proxy.

Onion Routing Background

A. Tor Relays and Proxies

For somebody to begin using the Tor network they first run a specialized proxy on their system. This proxy downloads the state of all Tor relays, which act similar to routers forwarding data. The proxy can then select a random path through relays to send an anonymous request. Every relay is connected to all other relays using TLS connections.

B. Tor Directory

The Tor directory contains a list of all active relays as well as information about their names, IP addresses, ports, onion keys and long term keys. This information would be used by both the relays and proxies in order to send packets between relays and encrypt/decrypt data sent/received.

Proxies can request information of all relays from the Tor directory to create circuits. Circuits are the paths through which data will flow before reaching the final destination. Once a circuit is created, the proxy will also use the onion keys provided to create the layers of the “onion” and encrypt the data recursively with each onion key.

C. Tor Encryptions

As mentioned previously, the term “onion” comes from the process of encrypting data sent between relays recursively with each onion key that belongs to a relay in the path. A proxy, once it has created a circuit for which to send data through, will encrypt a packet with the onion key of the destination proxy/relay to receive the message, and then with the second to last relay, and then with the third to last, and so on, finally encrypting it with the first relay. Once the data is sent, each relay will decrypt the message, read the cell header and then forward the packet with the appropriate logic.

D. Tor Cells

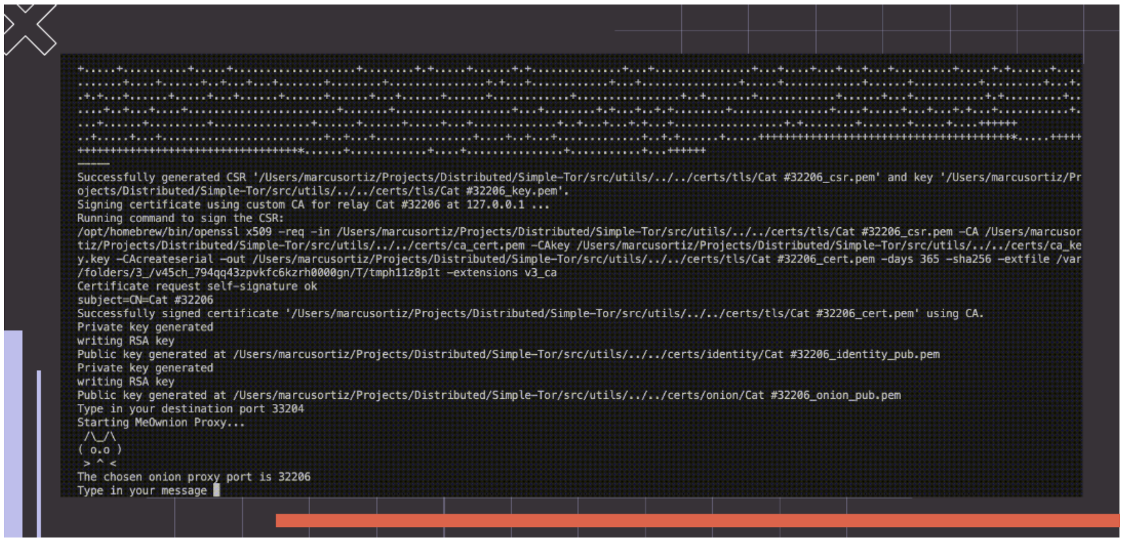
Tor cells refer to the headers used in sending packets to relays and help provide important metadata regarding the data. This includes the circuit id (an identifier for differentiating circuits) and the IP address and port associated with a relay.

II. Design and Implementation

A. Starting the Network - *start_tor.py*

To start up our project, the start_tor.py python file is ran using the terminal command "python3 start_tor.py". This will start a network of relays (represented by the OnionRelay python class) which will connect to each other. A proxy is also started (represented by the OnionProxy python class) through which a user can send messages to any relay on the network.

The proxy will start up a terminal GUI like the one below



```
+.....+..........+.....+..................+........+.+.....+......+.+..............+...+..............+...+....+...+...+...+.........+.....+.+......+....
.......+....+......+..+...+...+.......+........+...............+.+...+...........+...+...............+.....+...........+......+..........+.........+...+.
.+.+...+......+.....+...+.......+......+......+...+......+......+.............+.......................+..+.......+...............+.....+...+..........+.+.........+..
....+..+...+....+.....................+.......+.................+...+........+.+...+..+.+........+...............+....+.....+...+...+.+...+...........+.
...+......+.........+...............+......+......+......+...+.............+..+...+...+.+...+.................+.+........+.......+.....+....++++++
..+.....+...+........................+..+...+...............+....+..+...+.............+..+.+......+.....++++++++++++++++++++++++++++++++++++++*.....+++++
++++++++++++++++++++++++++++++++++*......+...........+....+...............+...........+...++++++
-----
Successfully generated CSR '/Users/marcusortiz/Projects/Distributed/Simple-Tor/src/utils/../../certs/tls/Cat #32206_csr.pem' and key '/Users/marcusortiz/Pr
ojects/Distributed/Simple-Tor/src/utils/../../certs/tls/Cat #32206_key.pem'.
Signing certificate using custom CA for relay Cat #32206 at 127.0.0.1 ...
Running command to sign the CSR:
/opt/homebrew/bin/openssl x509 -req -in /Users/marcusortiz/Projects/Distributed/Simple-Tor/src/utils/../../certs/tls/Cat #32206_csr.pem -CA /Users/marcusor
tiz/Projects/Distributed/Simple-Tor/src/utils/../../certs/ca_cert.pem -CAkey /Users/marcusortiz/Projects/Distributed/Simple-Tor/src/utils/../../certs/ca_ke
y.key -CAcreateserial -out /Users/marcusortiz/Projects/Distributed/Simple-Tor/src/utils/../../certs/tls/Cat #32206_cert.pem -days 365 -sha256 -extfile /var
/folders/3_/v45ch_794qq43zpvkfc6kzrh0000gn/T/tmph11z8p1t -extensions v3_ca
Certificate request self-signature ok
subject=CN=Cat #32206
Successfully signed certificate '/Users/marcusortiz/Projects/Distributed/Simple-Tor/src/utils/../../certs/tls/Cat #32206_cert.pem' using CA.
Private key generated
writing RSA key
Public key generated at /Users/marcusortiz/Projects/Distributed/Simple-Tor/src/utils/../../certs/identity/Cat #32206_identity_pub.pem
Private key generated
writing RSA key
Public key generated at /Users/marcusortiz/Projects/Distributed/Simple-Tor/src/utils/../../certs/onion/Cat #32206_onion_pub.pem
Type in your destination port 33204
Starting MeOwnion Proxy...
 /\_/\
( o.o )
 > ^ <
The chosen onion proxy port is 32206
Type in your message
```

Figure 1: The terminal GUI (cat included)

The proxy allows users to connect to the network and send any message they like (with a max message length of 256 bytes). It also allows users to specify the port that they would like to send the message to. In implementing the MeowOnion Proxy, we believed that accessing a TOR network has been popularized as a frightening and conspicuous activity, and so we wanted to model our program after a kind, charming creature such as a cat. This will allow users to feel more at ease with using this technology and embracing anonymity.

B. Advertising Relay States - *onion_directory.py*, *onion_directory_server.py*

The directory server acts as a central registry for the onion routing network, maintaining an up-to-date list of all active relays. This component is implemented across two main files: onion_directory.py, which handles the core directory functionality, and onion_directory_server.py, which provides the HTTPS interface.:

The directory server is built using Flask and implements three critical endpoints:

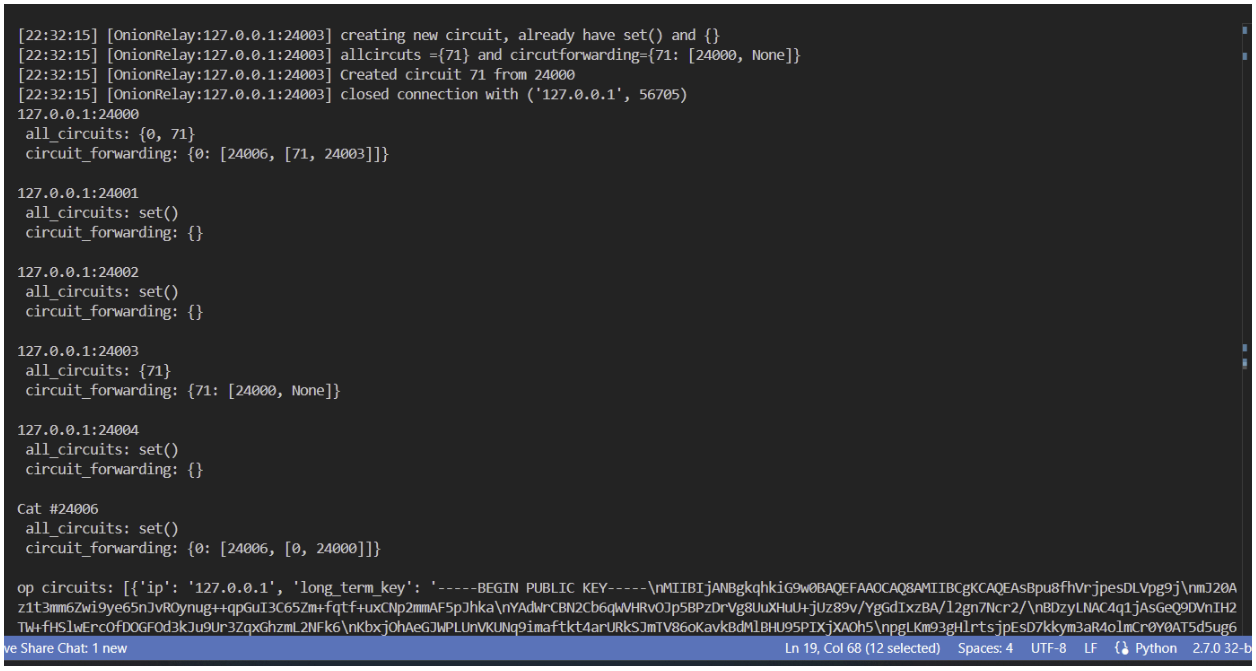
1. **/upload_state:** Allows relays to register themselves by providing
    - IP address and port
    - Public onion key for circuit encryption
    - Long-term public key for authentication
    - Cryptographic signature for verification
2. **/heartbeat:** Maintains relay availability through:
    - Periodic health checks form active relays
    - Signature verification to prevent spoofing
    - Automatic removal of inactive relays
3. **/download_states**: Enables clients to retrieve:
    - List of all active relays
    - Associated public keys
    - Network information for circuit building

C. Beginning an Anonymous Request - *onion_proxy.py*

The proxy creates a terminal GUI through which the user can send messages to other relays on the network. Once a user has entered a message, the process of constructing the packet will start, which includes creating the parameters for the circuit ID, source port, destination port and the data to send. Once the cell is created, through the TorHeader.py classes, the process to start a circuit commences.

D. Forwarding the Request - *onion_relay.py*

Once the message has been encrypted enough, the packet will be sent through the circuit of relays until it reaches its final destination. The circuit will be created through a series of CREATE and EXTEND and RELAY_DATA commands expressed in the cell of the packet. The CREATE command will be used to initiate connections between relays and establish relays with knowledge of the circuit. Each relay maintains a record of all circuitIDs it has seen aswell as the translation between circuitIDs and the port to forward the packet to. The EXTEND command will be forwarded by all relays with a forwarding port until it reaches the last relay in the circuit, at which point the last relay will send a CREATE message to the next relay to extend the circuit. The RELAY_DATA command simply relays a message until the end of the circuit.



```
[22:32:15] [OnionRelay:127.0.0.1:24003] creating new circuit, already have set() and {}
[22:32:15] [OnionRelay:127.0.0.1:24003] allcircuts ={71} and circutforwarding={71: [24000, None]}
[22:32:15] [OnionRelay:127.0.0.1:24003] Created circuit 71 from 24000
[22:32:15] [OnionRelay:127.0.0.1:24003] closed connection with ('127.0.0.1', 56705)
127.0.0.1:24000
 all_circuits: {0, 71}
 circuit_forwarding: {0: [24006, [71, 24003]]}

127.0.0.1:24001
 all_circuits: set()
 circuit_forwarding: {}

127.0.0.1:24002
 all_circuits: set()
 circuit_forwarding: {}

127.0.0.1:24003
 all_circuits: {71}
 circuit_forwarding: {71: [24000, None]}

127.0.0.1:24004
 all_circuits: set()
 circuit_forwarding: {}

Cat #24006
 all_circuits: set()
 circuit_forwarding: {0: [24006, [0, 24000]]}

op circuits: [{'ip': '127.0.0.1', 'long_term_key': '-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsBpu8fhVrjpesDLVpg9j\nmJ20A
z1t3mm6Zwi9ye65nJvROynug++qpGuI3C65Zm+fqtf+uxCNp2mmAF5pJhka\nYAdWrCBN2Cb6qWVHRvOJp5BPzDrVg8UuXHuU+jUz89v/YgGdIxzBA/l2gn7Ncr2/\nBDzyLNAC4q1jAsGeQ9DVnIH2
TW+fHSlwErcOfDOGFOd3kJu9Ur3ZqxGhzmL2NFk6\nKbxjOhAeGJWPLUnVKUNq9imaftkt4arURkSJmTV86oKavkBdMlBHU95PIXjXAOh5\npgLKm93gHlrtsjpEsD7kkym3aR4olmCr0Y0AT5d5ug6
```

Figure 2: The terminal logs displaying construction of the circuit

a. **Encrypting the Request - *tor_encryption.py***
The request is encrypted multiple times for each relay that will come into contact with the packet. At each relay, the relay will decrypt the message before forwarding the packet to the next hop.

E. Additional Features

a. **Certificates and Key Generation - *utils/certificates.py, certs/****
Each relay and proxy node uses an abstracted Certificates class to automatically generate private/public key pairs for their short-term onion key for encryptions and a long term identity key. In addition, to enable TLS connections a certificate is automatically created and trusted by a dummy CA . All of these files are stored in the certs/* directory, but most are auto deleted after running

b. **Unit and Integration Testing - *tests/*, test_directory.py***
To test individual parts and help debug the larger program various pytests were added such as TestCertificates which verifies the certificates and keys are generated and functional. Various tests also uses mocking to isolate.

c. **Logging - *utils/logging.py, logs/****

To debug threading and network reliant code, a specialized logging class was inherited in the Relay, Proxy, and Certificates classes which both stores and streams instance specific logs.

III. Simplifications, and Challenges

A. Ports as Nodes

Tor nodes (relays, proxies, directory) are represented using separate ports on a single device's localhost rather than separate devices. The logic is largely transferable as everything is still running on network sockets. This simplification helped greatly with testing and development as we only needed the current machine we were working on rather than multiple devices/VMs.

B. Flexible Connections

Relay connections to all other relays are not constantly maintained. This does not affect anonymity or any functionality since we restart sockets, only speed.

C. No TCP Streams

In our implementation, we chose to not account for TCP streams as the packets being sent between proxies/relays were assumed to consist of small ASCII messages and not multiplexed per proxy.

D. No TLS

Although the certificates are still automatically created, because of difficult debugging the project was switched to only use TCP connections

E. Simplified Keys

Rather than introducing Diffie Helman, the current implementation assumes we have symmetric keys available. It would not take a significant effort to extend to Diffie Hellman by building off the circuit create and extend circuit functions. In addition, our current infrastructure can already support asymmetric encryption if the relay's were to use their own private key, but was not chosen because of asymmetric size and speed limitations.

Appendix

Citations

- Tor: The Second-Generation Onion Router: https://courses.cs.duke.edu/fall24/compsci512/internal/readings/tor.pdf
- Python Sockets: https://docs.python.org/3/library/socket.html
- Python Threads: https://docs.python.org/3/library/threading.html

The Code

- https://github.com/alexiscruzdavid/Meownion-Proxy/tree/encryption

(Please use encryption branch)