Université Libre de Bruxelles
École polytechnique de Bruxelles
Master in Engineering Sciences
ELEC-H417

Communication networks : protocols and architectures
Tor Project

Authors
Amine Lafquiri
Jonathan Stefanov
Rayan Mokhtari
Gaspard Merten

23 December 2022

Professor
Jean Michel Dricot
Assistants
Verstraeten Denis
Wilson Daubry

TABLE OF CONTENTS

1 INTRODUCTION

For the project of ELEC-H417: *Communication networks: Architectures and protocols*, students were asked to implement a Tor network, which core principle is based on *onion routing*. This report will explain precisely how a Tor network works, and illustrate an implementation walkthrough of the network and an authentication protocol based on *challenge-response*.

2 PRINCIPLE OF TOR NETWORK

In a Tor network, messages are encapsulated with several layers of encryption. First of all, the client generates a path through the TOR network. Then it encrypts its message with the public key of each server in reverse order. The client will then send the message with multiple layers of encryption N to the first server. The server will *peel* one layer of encryption, parse the address of the next server and send the message encapsulated with N-1 layers of encryption to the next server until the message arrives at the final destination without any encryption. This method allows an anonymous connection since a server knows only the previous source of the encrypted message and its following destination of it. Once the last server receives the response of the actual destination server, it encrypts it using the symmetric key that was embedded in the message it first received (each symmetric key is generated by the client, encrypted using the public key of the relative server and used to encrypt the actual body of the message which is too lengthy to encrypt with a public key).
The goal of this project is a follow; to create a new implementation of the Tor Network which would enable a client to login himself onto an authentication server. Then using the token he just exchanged with a username/password pair, he should be able to make an authenticated request, retrieving information for his eyes only.

From a security point of view, the TOR network attempts to offer a certain degree of anonymity to the wider global internet. But it is not perfect. Indeed, if both the entry node and the exit node are compromised by the same malicious entity, the entity will know exactly what your IP is, to which server you sent a request and what the response was. To prevent such a thing from happening, a tor network should be composed of as many independent nodes as possible. When the client generates the path through the TOR network, it should try to pick servers in different countries or continents, increasing the difficulty for one entity to control the whole network.

One thing that was not disclosed until now is how the client retrieves the list of nodes. Indeed, the nodes communicate between themselves in a peer-to-peer fashion. But it would be too much of a load to force each of them to know all the other existing nodes (except maybe if a novel approach that could be based on a blockchain-like

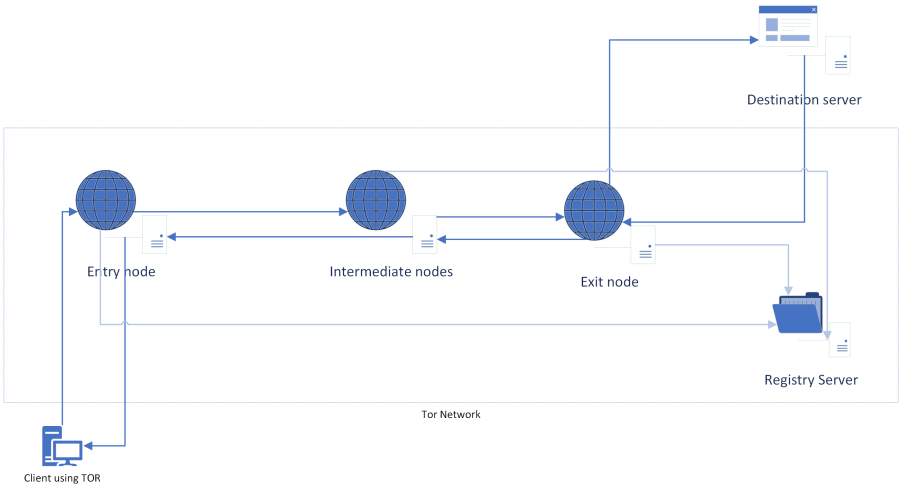

Figure 1: Graph of the tor network

technology was used). To that end, a centralized server called a registry node will maintain a list of all nodes that registered themselves to the registry. The registry will then, when the client asks for it, return the list of available nodes.

3 ARCHITECTURE

The python code is composed of multiple components which work together to create this secure network. Before looking at each unit in more depth, do note that this implementation of the TOR network is made over HTTP. Meaning that messages between the client and nodes of the network are transported in a secure fashion but over HTTP. In the end, this means that using the TOR Network would be like using HTTP(S) over TOR over HTTP.

3.1 Encoding

Before encrypting any messages, it is necessary to define how a message in the TOR network should look like. Since the network relays on HTTP to transmit messages between the client and nodes, the encoding of messages can be done in a more modern approach, indeed even JSON could have been used to communicate between each part of the network. But in order to optimize the size of the message, a proprietary protocol was used. It is as follows. The first character of the message can either be one if the node is the exit node or zero if it is not. If it is not an exit node, the IP address and the port of the next node shall be found. Finally, in either case, the actual body of the message is to be found.

$$\text{message} = (0|1)(\text{IP} : \text{PORT})?(\text{MESSAGE\_STRING})$$

3.2 Encryption

After encoding a message, it needs to be encrypted.
To encrypt a message of an undefined length, it is necessary (from a performance pov at the least), to first generate a symmetric key, encrypt it using a given public key and then encrypt the actual body of the message using that very same symmetric key.
On the other side, to decrypt a message, it is necessary to first extract the part which corresponds to the symmetric key. This can be easily done since the encrypted symmetric key possesses a fixed length. Once the symmetric key is decrypted, it should be used to decrypt the still-encrypted part of the message.
When a client makes a request through the TOR network, it conserves the list of symmetric keys it used to send the request. This allows it to decrypt the message once it is received. Indeed, each node encrypts the response it receives from its successor using the symmetric key found in the request it first received. If only the exit node encrypted the message, it would be easy for an observer to follow the path of a given packet since its content would be the same.

3.3 Tor Nodes

The Tor nodes are HTTP servers that are the main component of the TOR network. They can receive messages from any IP. Once a message is received, it is decrypted and decoded using the steps defined above. Based on the content of the decrypted message, it knows to which server next to send the request. In the end, the last node (called the exit node) will have the request totally decrypted and will make a request to the server that the client asked for. It will re-encrypt the response, again as described above, with the symmetric key it received with the initial request. This encrypted response will travel the inverse path until it reaches the client (being encrypted one more time at each hop). This way no node can know both the client who asked for the request and the content of the request at the same time.
To send a request through the TOR network using a node, the HTTP POST method shall be used.
Some specific endpoints are exposed over the global network such as the "public-key" endpoint which returns the PEM file corresponding to the public key of the node.

3.4 Registry Node

It acts as a centralized registry for every node of our Tor network.
It has endpoints which can return all the IP and public keys of all the nodes of the network, which will be used by the Tor client to create a secure path.
It has endpoints to add and delete nodes from the registry as well as an endpoint which

is able to check whether a node is still up.
One drawback of using this system is that this node centralizes our network a lot and is a single point of failure, it is not up to the client is not able to know the IP addresses of the Tor nodes and cannot form a secure path. This problem could be fixed by creating a decentralized network of Registry nodes which would communicate between themselves which endpoints they have and update themselves accordingly. This way, if one registry node is down we would still have a lot of other ones up.

3.5 Authentication Server

The authentication server is not an actual part of the TOR network, but it was implemented to demonstrate the capabilities of the TOR network to handle complex requests and responses.

1. /auth [POST]: requires a JSON body including a password and username keys. If the username and password correspond to the one it awaited, generate a unique token, store it and return it to the client in a JSON-encoded message (status 202). If it could not find the corresponding user, returns an empty response with status 404.
2. /private(.*) [GET]: requires the Token header to be present in the request. If the value of the header corresponds to a token stored on the server, returns the username of the user it is linked to. If the user is correctly authenticated it returns the 200 status code, if not the 403.
3. /(.*) [GET]: all other GET requests return 404.

4 INNOVATION AND CREATIVITY

4.1 TOR Over HTTP

For this project, since the TOR network was designed in a synchronous fashion and over HTTP, each message has to be understood by the exit node so that it can make the request. This means that instead of creating a tunnel allowing TCP traffic to flow through, the implementation proposed for this project works like an API. The client makes a POST request containing the full message, it is then transferred through the network until it reaches the exit node. That node parses the message and makes the request to the destination server itself. This may be restrictive since it implies that the exit node should be able to parse any kind of protocol but at the same time, it offers what could be called "security by design" since by parsing each message, it can also understand the content of the request and prevent sensitive information to leak to the

destination server.
Another argument in favour of building TOR over HTTP is that the code behind the network becomes way smaller and thus more accessible and maintainable. For this first iteration of the network, less than a thousand lines of code were used (everything combined).

4.2 Proxy

Because making requests using a terminal is not a very modern approach to browsing the internet, a proxy able to use the TOR network implemented in this project was created using the mitmproxy library. The proxy can be configured in any browser but keep in mind that only HTTP1.x requests will work. This is due to the limitation of the python-requests library. Indeed it does not support the HTTP2 protocol.

4.3 Demo

A superb demonstration script was built to demonstrate the capabilities of this TOR network over HTTP. Just launch the demo.py after installing the pip requirements of the project and wait until a message asking you to launch the demo shows up.

5 CHALLENGES

In our project, the group was dealing with a little problem. Indeed, since HTTP2 requests are not supported by the python-requests library, the network is able to carry the request but not make it. A possible alternative would be to use the HTTPX library instead of python-requests.

The network could also be reconfigured to act as a TCP tunnel if needed but still over HTTP. Only the client and the exit node should be a bit modified to support such a configuration. This would create a real tunnel between both ends but would result in no analysis/filtering being done on the traffic, especially if HTTPS is used.

6 CONCLUSION

In conclusion, the project was really interesting, as it allows us to create our own private network communication. We learnt about the world-famous system of communication TOR. We broaden our knowledge of communication networks by actually applying what is taught in the course. Our project was successfully implemented and we enjoyed every bit of implementing the network.