ECSE414 - Intro to Telecom Networks
Deliverable 1 - Project Description: Onion Routing: Group 10
Authors: C. Garcia La Rotta, D. Liu, E. Zhao, H. Wiltzer, J. Virdee, S.D. Mavroidakos, S. Mashaal
September 29, 2017

Project Description

The goal of this project is to implement and design an Onion Routing protocol, which ideally allows for users to communicate completely anonymously over the internet. Communication with Onion Routing involves peers on a network through which data is transmitted with multiple layers of encryption. These encryption layers get decrypted as the message is passed throughout the intermediate peers in the path. However, as opposed to proxy servers for example, the layered encryption ensures that no nodes along the message path are aware of who the sender was, so it is impossible to determine if a particular sender ever actually sent a message. Our Onion Routing system will support encryption on bidirectional communication to achieve this anonymity.

Performance of the system will also be assessed via simulation. A network topology which mimics characteristics of the internet will be used. Onion routing will be evaluated by comparing the lengths of routes taken by a packet using onion routing to one using shortest-path routing. Additional delay will also be included to calculate the total price of anonymity.

Methodology

The Onion Routing Protocol will be implemented with several independent routers (nodes) communicating via TCP packets. For all N nodes, each node n_i has an asymmetric cryptographic key pair $(pub_i, priv_i)$ and an IP address a_i. Additionally, a special server called the Directory Node contains the information $\{(a_i, pub_i) \forall i \in [\![1..N]\!]\}$.

Say a sender $\mathcal{S}$ wishes to send a request (and receive a response) from some destination $\mathcal{D}$. Firstly, $\mathcal{S}$ connects to the Directory Node and requests a path $\mathbf{P}$ through the network. The Directory Node then generates $\mathbf{P}$, a random and ordered subset of the onion-net's N nodes where $|\mathbf{P}| \leq N$. For each node $n_j \in \mathbf{P}$, the Directory Node negotiates using the corresponding pub_j and its own private key, thereby producing a symmetric key sym_j. The Directory Node then constructs a packet containing $\{(a_j, sym_j) \forall j \in [\![1..|\mathbf{P}|]\!]\}$ and encrypts it with the public key corresponding to $\mathcal{S}$, and sends the cipher to $\mathcal{S}$.

Now, $\mathcal{S}$ decrypts the cipher with his private key. He creates the request he wishes to send to $\mathcal{D}$, and encrypts with each sym_j successively, but in reverse order (so the first sym_j he encrypts with is $sym_{|\mathbf{P}|}$, which corresponds to the last node in the path). Before encrypting with each sym_j, a header indicating which node to send to next is prepended so that each node can determine where to send the packet after decrypting. $\mathcal{S}$ proceeds to send this many-layered "onion" of ciphers to the first node in the path.

Each intermediate node n_j behaves according to the exact same protocol. Upon receiving a packet, a node n_j decrypts the packet with its symmetric key to expose a_{j+1}, the address of the next node n_{j+1} in the path. After sending the packet to n_{j+1}, n_j waits for a response packet. When a response packet is received, n_j encrypts the response with sym_j and sends the packet to n_{j-1}. When $\mathcal{S}$ receives the response, he decrypts with each symmetric key successively (now in forward order), and can then read the response.

All nodes will be implemented as TCP servers programmed with the Python programming language using the Flask framework to handle concurrent requests. The Vagrant tool will be used to automate the creation of virtual machines which will simulate the nodes. Additionally, to save processing power, the

Directory Node will run locally on the sender's machine for the purposes of the simulation.

Furthermore, the NTRU cryptoscheme will be considered and investigated to provide key exchange for communicating the s_i keys, and can maintain security against quantum opponents.

Testing

The goal of the project is to achieve a minimalist implementation of onion routing and perform analysis of its performance via simulation. As such, testing of this project should be divided into two sections:
Verify and validate the onion routing implementation: We will follow a Test Driven Development (TDD) paradigm. Writing test cases first, then deriving the unit tests through equivalence partitioning and finally writing the necessary code for all the tests to pass. Throughout this iterative process each sub-team will rely on GitHub for the source code management tool. We will leverage GitHub's integrated bug tracking and ticketing system in order to equally subdivide the tasks among the group and to ensure regular code reviews are performed on the code base. Integration of TravisCI to our GitHub repository will enable us to perform Continuous Integration (CI) to ensure regression testing every time new functionalities or bug fixes are pushed to the master branch. With respect to the application testing, we shall focus on three main aspects of onion routing: *bidirectional communication*, *encryption of data through nodes* and *randomized path selection*. Each of these characteristics will be unit tested with virtual machines and after satisfying our requirements they will be undergo integration testing. In order to efficiently replicate the development and testing environment across the machines of each team member, Vagrant will be used to build and maintain the portable virtual development environments.

Analyze the performance of the application through simulation: There will be two tools that will be used, one is Nmap which provides a broader overview of the network and Wireshark to examine the packets transferred between nodes. We will be using Nmap for testing enumeration of network components and available application layer protocols at end hosts. It will also be used to identify any vulnerabilities in the network and to ensure the ports are connected to the correct hosts. Wireshark is used to test encryption requirements, provide an analysis of the chosen network and ensure contiguous nodes know the adjacent source destination and not the global ones. For further testing, we may use the mentioned tools such as Brite or Orbis to generate network topology to simulate characteristics in the internet.

Work Division

The work for this project will be divided into two main subteams, one of which will be the development subteam, and the other will be the testing subteam. As it stands, Camilo, Harley, Spiros-Daniel, and Stuart will make up the development subteam, and Dennis, Edward, and Jastaj will make up the testing subteam. It has been decided that each team member will document his own work, and Harley has volunteered to manage and compile the documentation that everyone writes. Furthermore, due to the difficulty of testing this system, Spiros-Daniel has volunteered to help out the testing team, so that the testing team can have assistance with testing out the more intricate parts of the system.

The development team will be in charge of developing the code and the software and network architectures. Furthermore, to follow a TDD paradigm, the development team will be in charge of writing unit tests for the features that they will be implementing.
The testing team will be in charge of conducting thorough integration and system tests that should reveal the behavior of the system under ordinary and stressful circumstances. The testing team will then report bugs or improvement ideas to the development team as they find them. Furthermore, the testing team will be in charge of setting up and managing the Continuous Integration of our software.