CS60 Final Project: Virtual Private Network

James Edwards, Rick Dionne, Willy Wolfe

June 3, 2017

1 Introduction

For our final project in Computer Networks, we built a Virtual Private Network (VPN) in Python. Our project consists of client and server programs which allow clients to connect to a server with a known address and communicate securely between clients over the network. A demo of the server is left running on port 5000 at `wolfe.cloudapp.net`, with public IP `40.79.72.247`.

2 Features

2.1 Core Features

Our project implements many common VPN features, including

- **Tunneling:** Packets sent to anything in the `10.10.0.x` address range are routed through the `tun0` interface, allowing client code to repackage them in a public IP packet to be sent to the server.
- **Private addresses:** Clients connecting to the server are given a private address with prefix `10.10.0` which serves as its identifier on the network.
- **Ping response:** Once a client has connected, the server will reply directly to pings (ICMP Echo-requests) with destination `10.10.0.1`.
- **Inter-network routing:** Packets sent to the server with the private address of another client as the destination will be routed to that client, allowing private connections between clients, including TCP connections such as SSH.

2.2 Extra Credit Features

- **Encryption:** All traffic to and from the server is encrypted using a shared secret, which is bitwise XORed with data in order to both encrypt and decrypt. This simple encryption scheme is a proof-of-concept for more heavyweight security schemes.

- **NAT (unfinished):** We attempted to implement NAT forwarding, such that the server would be able to act as a gateway to the public internet for connected clients. Unfortunately, issues with the routing tables on the client side and DNS resolution on the server side prevented us from realizing a fully functional implementation. These features are not included in the `master` branch, but can be found, along with documentation, in the `nat` branch on gitlab.

  At a high level, our approach to the NAT feature was to set the clients to route some or all of their traffic through the `tun0` interface, then to have the server take packets intended for the public internet, edit the source address and port to its own, and forward it through a raw socket, doing the reverse for responses received using internal data structures to map between public and private addresses. More details can be found at `https://gitlab.cs.dartmouth.edu/jgedwards/cs60project/blob/nat/CONFIGURATION.md`

3 Design

3.1 Client

The client program is divided into a number of files, each of which handles the logic for a specific subset of the configuration necessary to route properly packets through the `tun0` interface and return responses from the interface to the kernel.

- `vpncli.py`: The VPN client holds the core logic of the client, and initializes a client connection by opening a socket with the server, receiving a private IP address, and configuring the `tun0` interface with the assigned IP. The client then listens on both the socket and the `tun0` interface, handling read/write requests appropriately.

- `fakenet.py`, `pytun.py`: These two files handle the configuration of the `tun0` interface with the passed IP address, a set MAC address, and all other fields necessary for a functional interface.

3.2 Server

The server logic is entirely contained in one file: `vpnserv.py`. The server sets up a TCP/IP socket listening on a fixed port, and listens on that socket, accepting client connections and adding/removing them from the list of active connections as appropriate. Incoming packets are decrypted, interpreted, and responded to depending on their contents; Echo-requests for the server are replied to directly, packets intended for another active client are forwarded, and all other packets are ignored. The server can be closed manually by sending an `EOF` to its standard input, in which case it will close its active connections before exiting.

Encryption

Encryption logic is handled in `encryption.py`, which uses a shared secret to encode and decode all data, preserving privacy for clients.

4 Implementation and Limitations

4.1 Client

The client is implemented fully in Python 2.7, and requires the `scapy`[1] package, as well as `root` privileges in order to make changes to the `tun0` interface.

Once setup is complete, the client uses the `select` library to monitor both `tun0` and its socket with the server, and responds to data from both sources, encrypting and decrypting as necessary.

4.2 Server

The server is implemented in Python 2.7, and also requires the `scapy` package. The server maintains a list of open connections, which it monitors via `select`, and data structures mapping between private IP addresses and the socket of that client, to facilitate the correct routing of packets. Scapy is used to interpret packets and contruct ICMP echo-responses.

The server is limited to 254 simultaneous active connections, and will not accept new clients once this limit is reached. Clients which disconnect from the server and later reconnect will receive new private IPs; the old IP may be reused.

5 Testing

Testing was primarily done manually, with a goal of covering all areas of program logic. We successfully tested the following features:

- Pings between client and server
- Pings between clients
- TCP connections (SSH) between clients
- UDP connections (via Netcat) between clients.

6 Conclusion

Our project is a successful proof-of-concept for a usable, feature-complete layer 3 VPN, implemented entirely in Python. With more time and resources, our next steps would be to complete the NAT implementation and add more features to the VPN itself such as more robust encryption, authentication of users via certificates, and secure DNS resolution through a trusted server.

[1] `http://www.secdev.org/projects/scapy/`