Network Security (CS3403)
Project Report

WireGuard-Based VPN Communication System

Submitted by

Aditya Kushal

1RVU22CSE008

School of Computer Science

RV University

Submitted to

Prof. Sheba Pari

Assistant Professor

School of Computer Science

RV University

04/08/25

Abstract

This project demonstrates the creation of a secure virtual private network (VPN) using WireGuard, aimed at facilitating encrypted communication between a client and a server. The project includes key generation, secure tunnel setup, and a simple messaging system using Python sockets to test end-to-end encryption. Testing is also performed with both server and client running on the same machine.

Introduction

Project Background and Relevance

With the rise of IoT and Edge Computing, secure communication across untrusted networks is increasingly essential. VPNs offer encrypted tunnels to safeguard data from interception and manipulation. WireGuard, a modern VPN protocol, is lightweight, fast, and easy to configure, making it ideal for secure communication in edge environments.

Objectives

- Implement a WireGuard-based VPN.
- Enable encrypted client-server communication.
- Facilitate testing on the same machine.
- Validate the working of a secure tunnel using TCP sockets.

System Overview

System Architecture

The system consists of a WireGuard server and client, where encrypted keys are generated and configuration files are created for both sides. The data flow includes:

- Server listening on a TCP socket over the VPN tunnel.
- Client sending/receiving messages via the VPN.
- Secure packet flow over interface wg0 (server) and wg1 (client).

Design

Major Components:

1. **Key Generator**: Uses wg genkey and wg pubkey to generate private/public key pairs.
2. **Config Generator**: Dynamically writes config files (wg0.conf and wg1.conf) based on keys.
3. **Tunnel Initiator**: Uses wg-quick to start VPN interfaces.
4. **Messaging System**: Python scripts implementing server and client sockets over VPN IPs.

Interactions:

- The server binds to 10.0.0.1 and waits for client connection.
- Client connects to 10.0.0.1 via TCP and exchanges messages.
- All traffic passes securely through the WireGuard interface.

Security Features

- **Encryption**: All traffic is encrypted using Curve25519 key exchange.
- **NAT Routing and IP Masquerading**: Enables VPN clients to securely access the internet by routing traffic through the server while hiding their private IP addresses.
- **PersistentKeepalive**: Maintains session continuity.
- **Loopback Testing**: Ensures tunnel integrity by routing traffic to localhost via VPN.

System Requirements

- OS: Arch Linux (or any Linux with WireGuard support)
- Network interface (e.g., wlp3s0 for internet routing)
- Python 3.x
- Administrative privileges for WireGuard

Open-source Libraries and Tools

- **WireGuard Tools** (wireguard-tools): Used for key generation and VPN control.
- **Python 3 Socket Library**: Native library used to implement client-server messaging.
- **iptables**: For NAT routing and forwarding VPN packets.

Implementation and Testing

- Verified key generation and config creation via wg genkey and wg pubkey.
- Tested server-client communication using TCP sockets over the VPN tunnel.
- tcpdump was used to inspect traffic on loopback (lo) to confirm VPN encryption.
- Simultaneous server (wg0) and client (wg1) interfaces were brought up on the same device for testing.

Results

- Successfully established encrypted VPN communication using WireGuard.
- Implemented a client-server messaging system over the VPN.

- Verified encrypted UDP packets via tcpdump.
- System modularized into:
    - Key and config generation script
    - Client-server Python scripts

Future Extensions

- Add support for multiple clients.
- Implement a file transfer system over the VPN.
- Automate tunnel verification with additional monitoring scripts.

Conclusion

This project effectively demonstrated the use of WireGuard to create a secure communication tunnel between a server and a client. The hands-on implementation helped reinforce key networking and security concepts such as encryption, NAT, and packet forwarding. The modular Python-based system provided both flexibility and clarity for future experimentation or extension.

Screenshots

Messages without VPN

```
(base) [arch@arch CP3]$ python client.py
[Client] Connected to 0.0.0.0:5555
[You]: hello
```

```
(base) [arch@arch CP3]$ python server.py
[Server] Listening on 0.0.0.0:5555 ...
[Server] Connection from ('127.0.0.1', 38526)
[Client]: hello
[You]:
```

tcpdump without vpn

sudo tcpdump -i lo -n tcp port 5555 and "tcp[((tcp[12] & 0xf0) >> 2):1] != 0" -X

```
(base) [arch@arch CP3]$ sudo tcpdump -i lo -n tcp port 5555 and "tcp[((tcp[12] & 0xf0) >> 2):1] != 0" -X
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on lo, link-type EN10MB (Ethernet), snapshot length 262144 bytes
16:39:50.936068 IP 127.0.0.1.58274 > 127.0.0.1.5555: Flags [P.], seq 3636022212:3636022217, ack 1802198156, win 512, options [nop,nop,TS val 4176651690 ecr 4176512926], length 5
        0x0000:  4500 0039 fe25 4000 4006 3e97 7f00 0001  E..9.%@.@.>.....
        0x0010:  7f00 0001 e3a2 15b3 d8b9 4bc4 6b6b 5c8c  ..........K.kk\.
        0x0020:  8018 0200 fe2d 0000 0101 080a f8f2 a5aa  .....-..........
        0x0030:  f8f0 879e 6865 6c6c 6f                   ....hello
```

Messages with VPN

```
(base) [arch@arch CP3]$ python server_vpn.py
[Server] Listening on 10.0.0.1:5555 ...
[Server] Connection from ('10.0.0.1', 53122)
[Client]: hello
[You]:
```

```
(base) [arch@arch CP3]$ python client_vpn.py
[Client] Connected to 10.0.0.1:5555
[You]: hello
```

tcpdump with vpn

sudo tcpdump -i lo -n udp port 51820 -X

```
(base) [arch@arch CP3]$ sudo tcpdump -i lo -n udp port 51820 -X
[sudo] password for arch:
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on lo, link-type EN10MB (Ethernet), snapshot length 262144 bytes
22:33:22.216320 IP 127.0.0.1.60838 > 127.0.0.1.51820: UDP, length 112
        0x0000:  4500 008c 6bdf 0000 4011 1080 7f00 0001  E...k...@.......
        0x0010:  7f00 0001 eda6 ca6c 0078 fe8b 0400 0000  .......l.x......
        0x0020:  af96 e29f 1601 0000 0000 0000 2f6f 33ea  ............/o3.
        0x0030:  b5db 4b35 914a 4efb ab91 c2e6 c52f 7ce3  ..K5.JN....../|.
        0x0040:  b805 5e4d d1ce b48b 9188 250c 8d65 2d36  ..^M......%..e-6
        0x0050:  7941 e5ba 60d1 8bc2 f196 a845 f051 f23d  yA..`.......E.Q.=
        0x0060:  3b17 1909 a3cc 8a48 ca79 c4df 3935 f8df  ;.......H.y..95..
        0x0070:  6433 c8bd 1203 589f f15b bc1c 20a0 b393  d3....X..[......
        0x0080:  3d71 83fa da8f a8e1 6f3c d236            =q......o<.6
22:33:22.506048 IP 127.0.0.1.60838 > 127.0.0.1.51820: UDP, length 96
        0x0000:  4500 007c 6bfa 0000 4011 1075 7f00 0001  E..|k...@..u....
        0x0010:  7f00 0001 eda6 ca6c 0068 fe7b 0400 0000  .......l.h.{....
        0x0020:  af96 e29f 1701 0000 0000 0000 820e 50df  ..............P.
        0x0030:  6358 e5cc 30aa 3996 edb8 957e 4da3 d17e  cX..0.9....~M..~
        0x0040:  ca4f 36ef c2c1 f92e 9328 e37a 3f09 1cfd  .O6......(.z?...
        0x0050:  77c4 72ec 5196 5ba6 521e 5275 ab39 9336  w.r.Q.[.R.Ru.9.6
        0x0060:  1072 1344 db82 dee4 01cd a788 a5ef 0170  .r.D...........p
        0x0070:  e356 1381 ca5f e489 56fa 60a8            .V..._..V.`.
22:33:23.141036 IP 127.0.0.1.60838 > 127.0.0.1.51820: UDP, length 1312
        0x0000:  4500 053c 6cb7 0000 4011 0af8 7f00 0001  E..<l...@.......
        0x0010:  7f00 0001 eda6 ca6c 0528 033c 0400 0000  .......l.(.<....
```