Wireguard Connection Manager

Cameron Saifizadeh, Jonathan Huey, Sinclair Liang,
Sanjeet Jain, Kailas Krivanka

Abstract

The increased usage and integration of the internet for all tasks large and small has led to an increased concern over security. A Virtual Private Network (VPN) provides a secure connection for users to help address these concerns. Wireguard is a new VPN protocol that promises a smaller codebase and stronger performance. We have used this VPN protocol to develop our own connection manager microservice that will route users to servers and provide each user an isolated namespace. This ensures the security of a VPN connection while making configurations simple for customers.

Approach

The **connection manager** is a intermediary that passes the client's information to any of the Wireguard servers that it routes to. It then spins up a VPN interface that the client can use to establish a connection.

The Connection Manager receives information necessary to verify a user through a POST request made by a client. This information is then processed for any server permissions. If the user has no permissions, the Connection Manager will route to a generic server list using an algorithm to choose. Once a server is chosen, the manager generates a per session Wireguard key and sends a POST request to the server.

The **server** creates a new interface for the user. The connection provides an isolated environment for the user on the server using Docker containers. These containers are all run on separate ports so that multiple users may connect to a single server. Each server has a teardown protocol that will remove any interfaces that are no longer in use by a client.

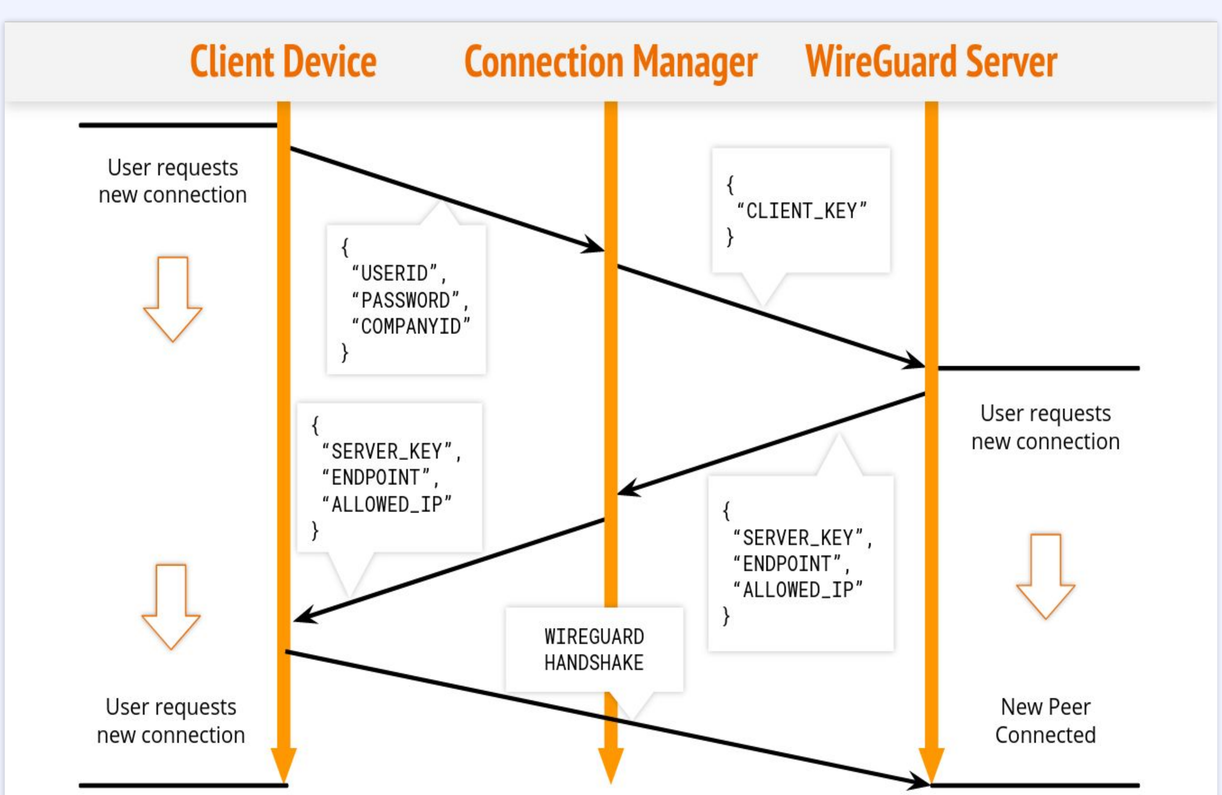
The **client** service can configure an interface using the response returned by the Connection Manager and make a connection to the server. This process streamlines a connection using only one command.

Overview

Cisco Umbrella is the division of Cisco focusing on cloud security. Their domain encompasses many products such as DNS-layer security, and a cloud delivered firewall. One of the products they provide is a cloud-based VPN service.

A VPN is a common network security construct that protects a user's traffic as it travels across the public internet to its destination. It acts as a trusted intermediary between a computer and the internet and securely encrypts all traffic sent between the user and the VPN endpoint. This prevents attacks such as traffic snooping and interception because all the user's traffic appears to be sent only to the VPN. Its actual destination or content cannot be determined.

We investigated Wireguard on behalf of Cisco Umbrella which hopes to develop a new VPN service based on this new protocol. We created a VPN connection manager that uses this new protocol as a prototype of this new architecture.



Special Thanks

Cisco Umbrella Team: Kyle Mestery, Greg Duraj, Adrian Oliver, Bryan Hong, Leo R. Augusto
CSE 115 Faculty: Richard Jullig, Arindam Sarma

Implementations

The main implementations that we focused on were Wireguard Go, Wireguard Rust, Wireguard Linux, and Boringtun. After extensively testing each implementation, we eventually decided to go with Wireguard Linux because it was the easiest and the most convenient to use. Also, each of the other implementations came with some issues.

Wireguard Rust implementation was incomplete and failed to build multiple times while running
Wireguard Go implementation had issues spinning up and running a connection.

Boringtun implementation was implemented successfully on the client. However, we chose Wireguard Linux due to our familiarity with it after using it for the VPS component

Results

We were able to successfully create a VPN connection manager for multiple simultaneous users that can route to multiple servers. This was tested with up to 3 servers. Users are authenticated with a username and password, have persistence in our database, and can use a single command to initiate a new Wireguard connection.

Moving Forward

We realized during the project that the architecture we created cannot run easily on commercial cloud platforms. This is because the VPS module requires kernel level access to networking APIs that are not always available in virtualized environments like Google Compute Engine.

Given more time to work on the project we would like to condense and combine some of our components and redesign the VPS service so that it could be deployed more easily.