Computer and Network Security
Virtual Private Network

By [REDACTED], 15008632

Table of Contents

Table of Figures

1.0 Introduction

1.1 Problem Scope

Virtual Private Networks have increased in popularity over recent years, with the rise of big data companies, targeted advertisements, and internet service providers having a constant look into your personal browsing it comes as no wonder that one would seek to improve the privacy of ones work or personal systems. In standard communication, sensitive data in transit such as meta data and traffic to internal HTTP services would be otherwise unencrypted and exposed to anyone who is sniffing around.

1.2 Aims

The aim of this project is to create an SSL Virtual Private Network client and server pair that can establish a tunnel between each other.
The explicit further goals of this are:

- Provide a graphic design to your VPN.
- Create a Host-to-Host tunnel using TUN/TAP.
- Create a Host-to-Host gateway tunnel.
- Create a Gateway-to-Gateway tunnel.
- Create a virtual private network.
- Authenticate both parties to the tunnel through the use of a certification authority and appropriate certificates.
- Implement the support for multiple VPN tunnels.

1.3 Structure

This report will follow a fairly linear structure, beginning with a review of literature contemporary to this project, this section is important to the later stages where the design and implementation choices will be determined by the knowledge garnered. Following this will be the methodology, this section will detail the chosen overarching methodology and detail the planned progression through the identified sections of design and development. After this will be the design section, where the architecture and structure of the VPN will be expressed explicitly. The next will be a comprehensive testing plan for the later testing section. After there will be a thorough review of what has been implemented along with figures and discussion of the explicitly identified key segments that are vital to the operation of the VPN created. The paper will then conclude with an evaluation of the entire project, including what has been done, what should have been done, and why they weren't done. Then in finality, the conclusion, which will summarize and explore the possibility of future work.

2.0 Literature Review

Recent years has seen an increase in the implementation of VPNs all over the world in various industries and even for small scale personal use. There is a growing market in commercial personal VPNs with the market being led by implementations such as Nord VPN and Express VPN, and many more on the way with Antivirus software beginning to package VPN software as well. Longsworth (2018) acknowledges some reasons for this rise in popularity, the first being the increase in technology trends such as the Internet of Things (IoT) and Bring Your Own Devices (BYOD). Furthermore, Longsworth identifies the changes in legislation as well, over the last ten years there has been a drastic shift in what data internet service provides should be gathering and keeping about its users, with some internet service providers being able to cancel plans and even issue cease and desist orders if you are accessing content they do not agree with.

Longsworth (2018) also identifies a clear tonal shift in the functionality of VPNs, when they first appeared they were used to provide secure remote access to network resources, but now they have changed to *"accommodate modern necessities"* which is what has led to privacy being the primary selling point of VPNs. Longsworth says that new VPNs will be experimenting with different forms of encryption in order to become more robust, as well as features like anonymous tokenised authentication or protocol obfuscation. Longsworth appears to be correct in his predictions, which presents no surprise given the rise of personal and business based VPNs, Longsworth even briefly mentions the impact this rise may have on future legislation, and how it may work to counteract the spread of VPNs.

Alshalan et al (2016) present a compilation of various virtual private network criteria, they base their study and criterion around a figure from a survey performed by Dimension Data (2014), which showed that *"79% of the 1600 surveyed IT and security professionals ranked mobility as top priority".* Throughout their paper, Alshalan et al (2016) identify the different practices of establishing tunnels, the layers of mobility, the pros and cons of security protocols, and their own constructed requirements list for a virtual private network. Almost all of the identified methodologies and choices are relevant to this study, and will be taken into consideration when identifying requirements. The only point made by Alshalan et al (2016) that is worthy of questioning is that *"Not all data needs to be encrypted",* which they identify by concluding that traffic that doesn't contain explicit data doesn't need to be encrypted and therefore can speed up the processing of the virtual private network both server side and client side. This is wrong on a fundamental level for two reasons, the first being that one cannot guarantee security on a platform that is willingly not encrypting communications whether the content is explicit or not, and secondly, there comes the issue of what is considered explicit and what isn't. Perhaps encrypting ping packets might be deemed a waste of time due to the fact it may not contain explicit, industry vital information, but an intercepting third party could use the data contained within to possibly get an idea of what the internal network looks like, it is not possible to deem certain data to be not-important and others important because the concept of importance is subjective.

Rossberg and Schaefer (2011) also present a comparison of various virtual private network technologies and methodologies. They compare the standard protocols such as IPSec, TLS, and PPTP, but what makes them differ from the previous study by Alshalan et al (2016) is that they also include considerations for the type of environment one might be deploying the VPN within, identifying Commercial and

Non-Governmental organisations, Government and Military VPNs, and Private Individuals. On top of this they identify the requirements for VPNs and their relevancy to different industries. This paper is extremely comprehensive, but it suffers due to the fact that it may be considered relatively outdated due to the shift of the industry.

Another study by Khanvilkar and Khokhar (2004) study various different implementations of virtual private networks available at the time of publishing, 2004. Whilst this study is relatively old in and perhaps considered outdated due to how much and how fast the industry is changing, its still worthwhile to explore what older virtual private network implementations looked like and how they worked. We do this in order to compare them to the current state of virtual private network implementations. Doing this we can see what the older implementations did wrong and what they did right, and use these as a basis for building our own VPN. In the paper, Khanvilkar and Khokhar (2004) present their findings in tabular format, which creates easy comparison between different implementations, they identify things such as whether it conforms to the contemporary Internet Standard, if it uses compression algorithms, what encryption or MAC algorithms they use, and the set-up complexity. In the paper they also compare the complexity of using various proprietary algorithms, and what has built-in support for routing.

An article by Caldwell (2012) reviews the effectiveness of VPNs and their various designs by referring to various industry experts. They also conducted a small survey targeting mid-market companies that reflects the 2012 mentality towards data security, with 59% of mid-market companies believing that investing in technology will facilitate data protection in its entirety, and only 1% of mid-market companies see data protection as the responsibility of everybody. The article presents some interesting flaws in the VPN structure, for example “VPNS secure the perimeter, but smartphones, laptops, and tablets physically cross the perimeter, and cloud applications store data outside of the perimeter”, which is essentially saying that whilst a virtual private network creates a virtual barrier, but is not able to secure the physical boundary or the pseudo-backdoor of devices that aren't complying to the companies standards.

Raghavan et al (2002) is another study which is relatively old but can be used for comparative purposes. It presents the requirements for a contemporary virtual private network implementation and compares technologies available at the time. It presents an interesting parallel by referring to Public Key Infrastructure as relatively up-and-coming in 2002 standards, whereas it is considered the go-to method by modern standards. If anything this parallel sends home the concept of ever changing methodologies, where Raghavan et al (2002) compared Public Key Infrastructure with Internet Key Exchange and found IKE to be the superior choice, the reverse is true in modern times. This creates the idea that whilst we consider PKI to be the obvious go-to now, will it remain so into the future? Is the sustainability and longevity of PKI based implementations reduced due to the transient nature of IT technologies? In a similar manner, the paper refers to SHA-2 as up-and-coming, a companion to the newest encryption standard, Advanced Encryption Standard (AES).

3.0 Requirements

The NCSC have supplied numerous guidelines for those who would be developing or implementing virtual private networks into systems, detailing what should be used and why. This section will briefly cover the design choices for this project in these identified areas.

3.1 Protocols

There are two main protocols for VPN implementations, Transport Layer Security (TLS) which operates with a third party client and server, is standardised in a variety of remote function calls, and is generally more reliable when traversing Network Address Translation (NAT) devices or enterprise firewalls. The problem with TLS is that firstly, different products and implementations will rarely be able to interact, and while TLS is standardised in remote function calls, how to use said remote function calls is not standardised, often leading to some confusion.

The other generally used protocol is IPSec, which is open standard, promotes interoperability, and requires no additional products. IPSec does however suffer from a key issue, this being that some networks will restrict IPSec traffic, often due to architectural issues with firewalls not allowing required protocols ISAKMP (port 500), ESP (IP Protocol 50), and AH (IP Protocol 51) or the firewall not handling fragmented IPSec packets and not replying to ICMP-Unreachable packets and therefore breaking path MTU detection (Carmouche, 2006). Due to the nature of this project we will be utilising TLS, but IPSec may be explored if there is time. We will be using TLS simply because of the reliability it possesses with traversing NAT devices and firewalls, given that our testing environment will be virtual machines with a NAT configuration.

3.2 Cryptography

When it comes to encryption over a VPN tunnel you have two distinct options. The first is key-sharing where you will encounter the difficulty of supplying each client with its key securely. The alternative is client certificates, which possess numerous benefits over conventional key sharing methods, including the ability to revoke a single certificate on the network and store private keys securely in a trusted platform module or trusted execution environment. In this implementation we will be utilising digital certificates in order to encrypt and provide authentication, these would usually have to be provided by a third party certifying body but for the sake of this project it has been deemed acceptable to use self signed certificates but it must be acknowledged that in the case of a industry implementation certificates would be required to be from one of these bodies.

3.3 Utilization Policy

There are two ways an organisation can allow its employees to utilise a VPN, Forced or optional. Using the forced configuration will push all traffic from the machine through the VPN tunnel, and no user traffic can transit

outside of that connection. The VPN client is required to enforce this, or the clients firewall, the two can be used to enforce the policy in conjunction. This obviously comes with the downside of if the VPN client cannot connect then absolutely no network activity from the device will be possible until the VPN is disabled. This is particularly evident in the fact that Forced VPNs can create incompatibilities with captive portals on public WiFi unless the platform offers a way of authenticating them. Making the VPN optional allows users to disable the VPN and evade protective monitoring and auditing systems, which increases the risk of being attacked over the network and of users circumventing corporate policy restrictions. In the context of an organisation a Forced VPN policy would be desirable despite its drawbacks, if anything the drawbacks mentioned may actually contribute to increasing security anyhow. In the context of this project however it will be possible to drop the VPN connection at will simply for the ease of testing and debugging.

3.4 Connection Type

VPNs need to be established in order to provide benefits, and should remain connected whilst in operation then reconnect when connection is lost. There are typically three ways of implementing this, Automatic, Triggered, and Manual. Automatic VPN connections are created whenever the device has software request network connection, Triggered connections are created when certain network connections from a predefined list are made and Manual requires the user to initiate the VPN by explicitly deciding to take action, the user will have to reconnect if the connection drops. The way in which these connections are defined by the NCSC gives some pause, as in theory you could have a manual connection that re-establishes on connection loss simply. Therefore it should be assumed what the NCSC detail here is the method of creating the first connection, and any concept of reconnecting is a separate functionality. In this implementation the initial connection method will be manual, but the client will attempt to reconnect if connection is lost.

3.5 Full-Device or Per-App

A Full-Device VPN will require all of the programs and processes running on the machine to be transmitted via the VPN tunnel, this promotes additional security by limiting the possibility of traffic being intercepted. Per-App allows for the configuration of certain programs and processes using the VPN tunnel and other not. This is good in cases where an organisation is utilising latency sensitive machines such as in the human machine interface in industrial control systems. Per-App implementations also allow for the specific blacklisting of apps, which may prevent unwanted personal apps accessing the network in the case of bring your own device setups. In theory the optimal would be to create a Full-Device configuration, and that may be achievable in the case of this project. However, at first the VPN implementation being created will work on a pseudo Per-App basis in the sense of we will be manually routing IP addresses, this may change depending on the overall progress of the project.

3.6 Split Tunnelling

A method of having some traffic utilise the VPN whilst other traffic is permitted direct access, generally achieved through defining certain network routes as available to the VPN. It is generally utilised when the VPN is providing access to internal services on a corporate network. This pairs well with Per-App VPNs, with latency sensitive actions being permitted to exist outside of the tunnel rather than being forced to use and then connecting directly to the internal service. Split tunnelling is generally not support natively on clients, Windows operating system can be configured to split tunnel by using custom routing rules. Clients built into devices like smart phones and tablets rarely support this. Enabling split tunnelling increases the risk that sensitive data will be exposed to an unprotected network and could enable external attackers to access resources by pivoting through. In a context where you are relying on the VPN to route traffic for protective monitoring or duty of care reasons, enabling split tunnelling will undermine the benefit gained from these protections. The base form of this project will not implement split tunnelling explicitly, but if there is extra time attempting to implement such a functionality would be desirable.

3.7 Captive Portals

Login portals are present on some public WiFi networks, often when attempting to connect to these the device will be redirected to a webpage where one is required to fulfil certain requirements, these are captive portals and they can interact strangely with VPNs. There are two methods of improving connectivity with the captive portals, the first being to simply disable the VPN, manually navigate and connect, then re-establish the VPN. However the VPN can detect the presence of a captive portal and present the user with a helper application to authenticate before re-establishing the VPN. Some platforms, such as iOS and macOS have this helper built in and are otherwise available as third party software on other systems. The use of helper applications is generally preferred if one must use a public WiFi connection. In this implementation we do not expect to test interaction with captive portals due to the static nature of the VPN we are developing.

3.8 Miscellaneous Considerations

There are a few minor considerations that are required to be addressed in the context of VPN implementations. The first is client and gateway security updates, both the client and the server will be ran on Linux based operating systems that will update to the latest security patch as it is available. Secondly, tethering devices with a VPN is not within the scope of the project and therefore not tested or optimised.

4.0 Methodology

This section will explore the development approach and cycle in regards to the creation of the virtual private network programs. It will begin by comparing varying methodologies; proceed into outlining the approach and development in the

context of our chosen methodology; then finalize with a brief description of each step.

4.1 Varying Methodologies

Due to the continuous nature of this project, it would be difficult to build the virtual private network with a Waterfall-like methodology. Waterfall is usually built around a clear set of goals and development aims with a set date by which the project will be considered finished and deployed. Our project has varying objectives, some of which are achievable only after previous ones have been accomplished. The virtual private network project has many different possible requirements, all of which are in some way desirable, and in cases where one might have to pick between one functionality or the other, given additional times it would be sensible to implement the one not picked at a later date, this way we can achieve a broad range of functionalities. Given this, the best method for our development would of course be an Agile approach. This way we can separate our critical functionalities into the initial sprints and then append additional functionalities within the continuous development part of an Agile methodology. Within each increment we plan for, there will be room for research, planning, development, and testing. Figure 1 is the brief plan for the cycle of this project.

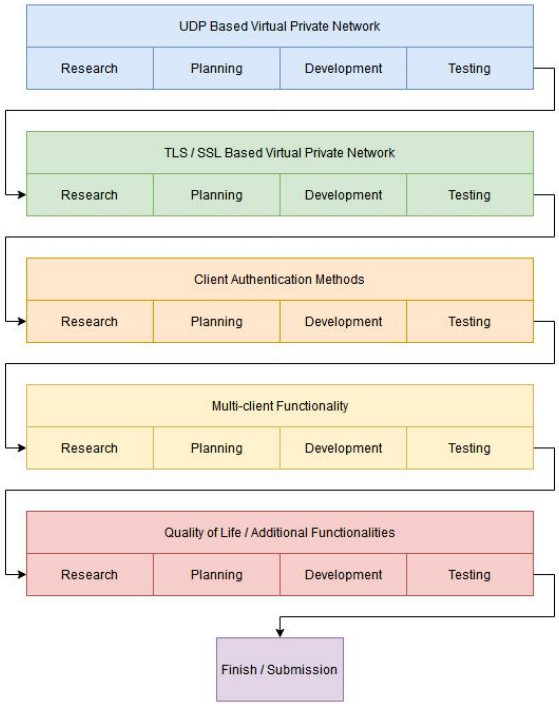

Figure 1 - Brief Development Cycle Plan

4.2 Steps

The first step is to develop the virtual private network within the context of the User Datagram protocol. Doing this is fairly simplistic, but acts as a solid foundation and a good way to learn the basic concepts behind that which will be used in more complex contexts later. Such concepts are the network layout and the TUN/TAP functionalities, more of which can be seen in 5.1 System Architecture.

The second step is implementing the virtual private network in a Transmission Control Protocol context, alongside this we will be using Transport Layer Security protocols in order to allow the client to verify the server, specifically using certificate based public key infrastructure to allow a communication between the two. This functionality will be implemented on top of the existing TUN/TAP functionality, and will not necessarily replace the UDP but act as an offer of a more secure communication method.

The third step is to authenticate the client, despite sounding like it would be a shorter step than the previous, there is much within this development stage that is open to consideration, and it is possible that numerous methods will be explored in full.

This leads in the fourth step, Multi-client functionality, which will allow the server to handle multiple connections and route the data and packets accordingly whilst also optimising how the server operates in the context of what it processes so that no communication is left unprocessed indefinitely. With this step concludes the acknowledged critical functionalities, and so begins the continuous development section of the project. Whilst Figure 1 details the fifth step as a single section, it actually takes place across numerous steps and could include numerous different functionalities. The goal overall would be to keep adding functionalities until it is not possible any more.

5.0 Design

This section will detail and display the various design choices in regards to the creation of the virtual private network, this will include all visualisations created in order to assist in the implementation of protocols and functionalities. One of the goals for the implementation of the VPN is to not separate each of the functionalities into different programs, but rather offer all of these different functionalities within a single demonstrable program. Given this, the design section will detail absolutely all functionalities and protocols that were used in all contexts, even if they are eventually invalidated by a subjectively (or objectively) superior implementation, for example despite the fact the implementation utilises primarily TCP/TLS the UDP part of the project is still available and will still be explored in detail.

5.1 System and Network Architecture

5.1.1 Networking

In order to ease development, the implementation will be created across a series of virtual machines with network adaptors that will emulate the context of two geographically separate networks, with one being a single machine trying to access another network with internal only machines. Virtual Box offers an easy way of creating this emulation, the two geographically separate, forwarding facing machines will emulate being connected to the Internet using a NatNetwork adaptor, one machine being the client and the other being the server. The internal machine will have a single network adaptor, a Host-Only adaptor, which will allow to to connect to other machines using a similar adaptor but not to the internet. In this way, there is no way for the client machine to access the internal machine without going through the server machine, which possesses a second Host-Only network

adaptor allowing it to communicate with the internal machine.

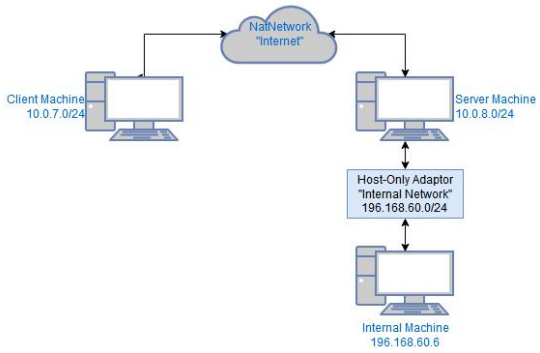

Figure 2 - Development Environment Network Layout

In the event that development proceeds as desired, it is possible that the network joining functionality is implemented, where both the server and client act as gateways on the virtual private network allowing internal machines within both that can access internal machines in the opposing network. In this context, the layout will be similar, except the client will also possess a Host-Only adaptor and an adjoining internal machine, as seen in figure 3.

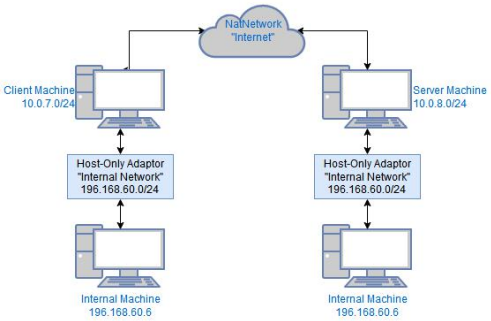

Figure 3 - Dual Gateway Network Layout

5.1.2 TUN / TAP

The TUN/TAP interface is what will be used to bridge the gap between the two Internet facing machines. Using this, two geographically separate networks can be joined into a single network, allowing access through the gateway machine hosting the server application. Each TUN interface will be assigned an IP address, all traffic coming from to going to the tunnel will be routed through the TUN interfaces. In the event of Multi-client functionality, there will be numerous tunnel application TUN interfaces on the server side, for the sake of clarity these will be left out of the diagrams.

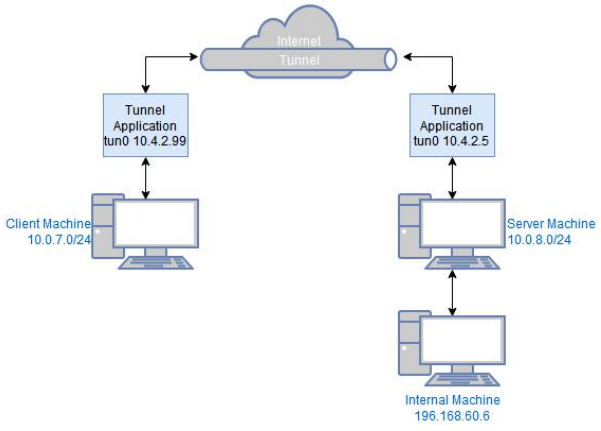

Figure 4 - Network with TUN Interfaces

TUN/TAP is a virtual interface that allows user space programs to directly interact with the process of data transmission, unlike physical interfaces that connect computers to other physical interfaces, virtual interfaces connect computers to these user space programs. Virtual interfaces that wish to route across networks obviously have to eventually interact with physical interfaces, the interaction between physical and virtual interfaces as well as their respective layers is viewable in Figure 5.

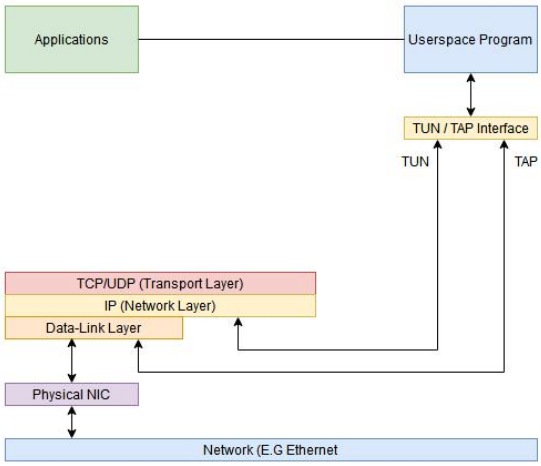

Figure 5 - Physical and Virtual Interface Interaction

5.1.3 User Datagram Protocol (UDP)

The first communication protocol utilised is the User Datagram Protocol. It is easy to implement but as a result comes with a lot of drawbacks. UDP is considered unreliable and unordered, when two packets are sent it is not possible to truly predict what order they will arrive in, and that is if the packets arrive at all. UDP has no concept of acknowledgement, and therefore it is not possible to coordinate transmission or retransmission. We start with UDP simply because of its simplistic nature and doing so we have a strong base from which we can implement other 'superior' protocols. Whilst the UDP functionality will remain relatively untouched after proceeding into the second step, it is possible that during the fifth step there will be a chance to implement security over UDP using Datagram Transport Layer Security (DTLS). The operation of DTLS can be viewed within the TLS / DTLS section.

5.1.4 Transmission Control Protocol (TCP)

After having established and tested the user datagram implementation, one can move onto the TCP implementation. TCP is generally more desirable than UDP for many reasons. TCP is significantly more reliable than UDP due to the fact that it implements the concept of confirming a received packet, if this confirmation is never received then it will work to retransmit the packet. Furthermore, TCP is ordered, packets transmitted in a certain order will arrive in that order. To present additional reliability, using checksums TCP allows for error checking and tampering within its packets. Implementing TCP within our VPN allows for the implementation of Transport Layer / Socket Layer Security, which will provide us with a convenient way to authenticate the server and coordinate the securing and encryption of data.

5.1.5 TLS/SSL/DTLS

Transport Layer Security provides a secure channel over which two applications can communicate, the data within which is private and its integrity is preserved. When a channel is secured using TLS you can guarantee the communications confidentiality, integrity, and the authenticity of the communicating applications. TLS and its functionalities sit between the application and transport layer as shown below.

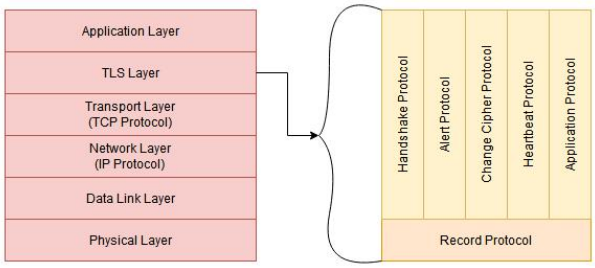

Figure 6 - TLS Layer Position

Unprotected data from an application is given to the TLS layer, which handles encryption, decryption, and integrity checks. The flow of our implementation when utilising TLS is visualised in Figure 7.

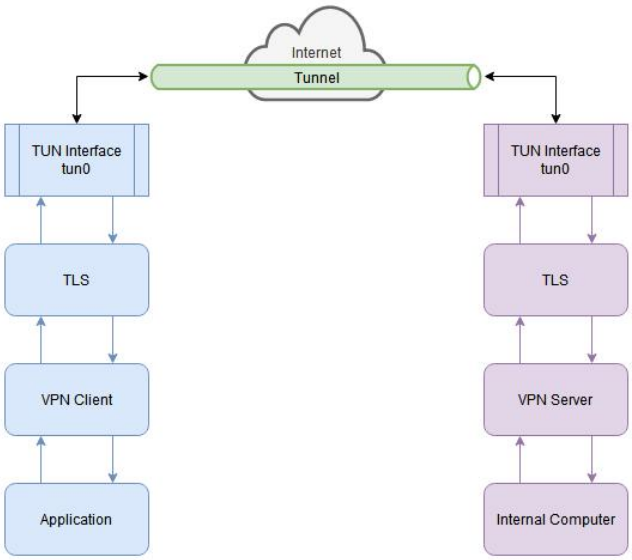

Figure 7 - VPN Flow with TLS

5.2 Object Oriented Design

Within this project the goal is to maintain a fairly modular design in regards to the various functionalities within the implementation. This meaning in essence, the developer should be able to choose functions and characteristics as they see fit to the context to which the implementation should be retrofitted for. For example, some may consider question the use of User Datagram Protocol functionality when Transmission Control Protocol is available, but we will continue the research and development into UDP functionalities and security simply for the sake that there may be somebody within the certain context that could make use of it. Therefore the plan is to separate the implementation into five different components, the first two being the obvious main and header, then the remaining three being TCP/TLS, UDP/DTLS, and miscellaneous functions.

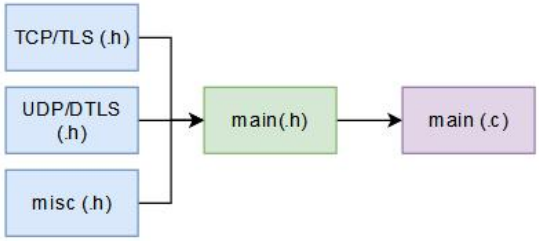

Figure 8 - Object Oriented Relations

Whilst the contents of TCP/TLS and UDP/DTLS my already be obvious from previous sections, the third miscellaneous file exists simply to house quality of life functions that can be used in both of the former files and do not necessarily belong specifically in one or the other. In essence the miscellaneous header exists to remove redundancy from these two files. The three modules will be compiled into the main header which will then be compiled onto the main.c file. All of the header files will be programmed to use header guards in order to prevent object collisions.

6.0 Test Plan

Throughout development the concepts of confidentiality, integrity, and availability will always be kept at the forefront of consideration. This characteristic will be inherited into the testing as well, meaning that each of the test cases within each rung of development will be use the CIA concepts personalised to the particular use case. Below we will outline the conceptualized test cases based on previously identified increments.

6.1 UDP / DTLS

6.1.1 Confidentiality

The first sprint of development is fairly limited to simply implementing a UDP mode of working to understand underlying concepts. Confidentiality in this case relies on the level of user access when connecting to the VPN, therefore theoretically the VPN server should handle who can access the VPN based on credentials passed. In the initial implementation this will most certainly not be the case, as it will require the implementation of DTLS to be able to function genuinely. Despite this, we will still test at this phase, if only to demonstrate what needs to be improved upon. The tests will be fairly simplistic, a test to acknowledge whether the authentication of users exists or not would suffice.

6.1.2 Integrity

Once again, the characteristic of Integrity in the context of UDP also requires the presence of DTLS, which will not be present at the beginning of development. We will repeat what has been established in the previous section and attempt to test for integrity anyway, if simply to demonstrate what needs to be improved upon. To test this we will peer into the packet on both sides of the tunnel using WireShark and demonstrate that the packet is in fact not secure.

6.1.3 Availability

Availability in this increment is perhaps the only characteristic that can be properly tested. In reality availability will be dependent on whether the client and server can run effectively on their respective machines and efficiently transmit, receive, and redirect. In this case we plan to perform some longevity testing, essentially leaving the setup running with the client machine constantly pinging the internal machine. The percentile of packets lost will determine our levels of success, 0% packet loss being the obvious best case, 1-5% being optimal, 6-10% being manageable, and 11% plus being unacceptable. Within this section we will also test how the client and server manage the downtime of one another. The predicted case for this is that the server will be unaffected by the clients downtime, but the client will be drastically affected by the downtime of the server; whilst this seems obvious, the purpose of this is to gauge what is require should one of the two lose connection. If the client loses connection, the server can remain as it is and the client can reconnect; but if the server goes down, both the client and the server will need to be restarted, etc.

6.2 TCP / TLS

6.2.1 Confidentiality

This characteristic will be implemented at this stage to allow for the client to authenticate the server using the TLS certificate architecture, the process of authenticating the client server-side will come in the next increment. Within this section we will be testing for successful TLS handshakes, which will be demonstrated with WireShark, this will demonstrate the TLS fundamentals as well as the client-side authentication of the server. Similarly to the previous sections, we will also test for the absence of server-side authentication.

6.2.2 Integrity

The security of our data in transit will be implemented with the implementation of TLS, which will allow us to utilise a common encryption method across the client and server, meaning that communications will be secure and integrity can be assured. Once again this will be tested and evidenced using WireShark.

6.2.3 Availability

Similar to testing availability in the UDP section, we will be testing longevity by running a constant stream of pings for set intervals in order to evidence what percentile of packet loss our implementation is operating in, we will still be utilising the margins identified previous. We will also replicate the shut off tests, expecting similar results depending on the state of the TCP/TLS implementation. In theory TCP should allow for timeout-based retransmission, but whether this will work effectively in the context or will require a more complex implementation remains to be seen, we will still test for its presence however.

6.3 User Authentication

6.3.1 Confidentiality

As identified in the previous section, this characteristic will be focused on the concept of server-side authentication of the client, so that only those who are supposed to be logging into the VPN are able to and those who are not permitted are denied. Within this section we will obviously test the consistency and reliability of the methods implemented, but we will also be testing that they work alongside the client-side authentication of the server, all of which will be evidenced.

6.3.2 Integrity

Technically the authentication should occur after the TLS handshake, this way we can assure that all user data being exchanged, even if wrong, is still secured by a mutually agreed cryptographic method. The testing taking place here will be to analyse the contents of the packets sent to see if we can determine any useful user information from them, these tests will only be considered passed if the conclusion is that we can gleam absolutely nothing.

6.3.3 Availability

Relatively simplistic, we will be testing that the authentication methods causes no downtime during the main operations, this shouldn't be the case at all. The only delay created by the User Authentication should be the extra time needed to accommodate for the User Authentication, which if implemented correctly should be negligible.

6.4 Multi-client Operations

6.4.1 Confidentiality

The characteristic of confidentiality in this context should be inherited from the implementation of confidentiality in the previous contexts. We will still test that each client can successfully and consistently authenticate the server and the server can authenticate the multiple clients. If in this section the multi-client functionality is implemented in UDP, we cannot guarantee confidentiality until DTLS is worked in as well, if this is the case, we will still test for and evidence its absence.

6.4.2 Integrity

Once again, integrity should be inherited from the previous sections. We will continue to test that the traffic between multiple clients and servers is encrypted. If implemented we will also demonstrate if the communications between the multiple UDP clients and server are encrypted or not.

6.4.3 Availability

This characteristic will be tested similarly to how it has before but on a larger scale. The test will be similar for both the TCP and UDP implementations. The network will be setup to test traffic coming from multiple clients for an extended period of time, we will also attempt to increase the number of clients but the extent of this test cannot be predicted due to hardware limitations

6.5 Further Development

Due to the fact that it is not necessarily known what will be developed in this section, since its contents will be focused around limitations identified in previous rounds of testing, the rationale for testing in these sections will be detailed in their respective sections in 7.0 if undertaken.

7.0 Implementation and Testing

7.1 UDP

7.1.1 Implementation

The implementation of UDP is fairly standard, it begins with the creation of a TUN device on both client and server, then the two split in order of operation with the server creating a UDP server and the client attempting to connect to a server using the details we have supplied it.

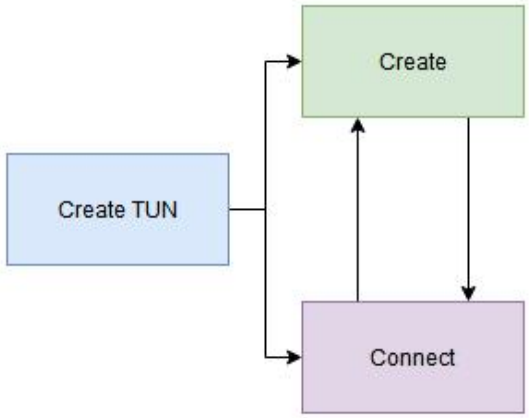

Figure 9 - UDP Basic Flow

In its current state, the server creation is required to occur first, there is a small window in which the client can be initialised first and still connect to the server, but it is unreliable and therefore the server should be initialised first to avoid soft-locking both.

The only notable deviations from the standard UDP implementation has been the inclusion of a method of configuring the TUN and routing table after the creation of the TUN interface, this occurs in both client and server. The server also includes the command to enable packet forwarding in case the host does not already have it enabled.

```
system("sudo sysctl -w net.ipv4.ip_forward=1");
int tunfd, sockfd;
tunfd = createTUNDevice();
system("sudo ifconfig tun0 10.4.2.5/24 up");
system("sudo route add -net 10.4.2.0/24 tun0");
sockfd = initUDPServer();
```

Figure 10 - Server System Calls

```
int tunfd, sockfd;
tunfd = createTUNDevice();
system("sudo ifconfig tun0 10.4.2.99/24 up");
system("sudo route add -net 10.4.2.0/24 tun0");
system("sudo route add -net 192.168.60.0/24 tun0");
sockfd = connectToUDPServer();
```

Figure 11 - Client System Calls

In the future it may be prudent to research a way to make the routing calls dynamic in nature to prevent the hard-coding of an IP address.

7.1.2 Testing

7.1.2.1 Confidentiality / Integrity

Source	Destination	Protocol	Length	Info
10.4.2.99	192.168.60.3	ICMP	100	Echo (ping) request
10.0.2.4	10.0.2.5	UDP	128	43621 → 55555 Len=84
10.0.2.5	10.0.2.4	UDP	128	55555 → 43621 Len=84
192.168.60.3	10.4.2.99	ICMP	100	Echo (ping) reply

Figure 12 - Ping from client to server

Figure 12 shows the ping packet in its journey to and from the server as seen by WireShark running on the client machine. When accessing the UDP packets that allow the transit of the packet, you can observe clearly the contents of and discern the original ping packet payload. Figure 13 shows the ping packet, figure 14 and 15 show the Ping packet clearly locatable in the UDP packet going both ways.

```
0000  00 04 ff fe 00 00 00 00  00 00 00 00 00 00 08 00
0010  45 00 00 54 e7 31 40 00  40 01 4a 65 0a 04 02 63
0020  c0 a8 3c 03 08 00 54 57  1d 56 00 01 44 b8 88 5e
0030  ce 37 00 00 08 09 0a 0b  0c 0d 0e 0f 10 11 12 13
0040  14 15 16 17 18 19 1a 1b  1c 1d 1e 1f 20 21 22 23
0050  24 25 26 27 28 29 2a 2b  2c 2d 2e 2f 30 31 32 33
0060  34 35 36 37
```

Figure 13 - Ping Packet Content

```
0000  00 04 00 01 00 06 08 00  27 60 51 63 00 00 08 00
0010  45 00 00 70 99 ed 40 00  40 11 88 87 0a 00 02 04
0020  0a 00 02 05 aa 65 d9 03  00 5c 18 76 45 00 00 54
0030  e7 31 40 00 40 01 4a 65  0a 04 02 63 c0 a8 3c 03
0040  08 00 54 57 1d 56 00 01  44 b8 88 5e ce 37 00 00
0050  08 09 0a 0b 0c 0d 0e 0f  10 11 12 13 14 15 16 17
0060  18 19 1a 1b 1c 1d 1e 1f  20 21 22 23 24 25 26 27
0070  28 29 2a 2b 2c 2d 2e 2f  30 31 32 33 34 35 36 37
```

Figure 14 - Ping Packet UDP Transit Client -> Server

```
0000  00 00 00 01 00 06 08 00  27 60 f0 ba 00 00 08 00
0010  45 00 00 70 b2 2a 40 00  40 11 70 4a 0a 00 02 05
0020  0a 00 02 04 d9 03 aa 65  00 5c 63 c4 45 00 00 54
0030  3e 75 00 00 3f 01 34 22  c0 a8 3c 03 0a 04 02 63
0040  00 00 5c 57 1d 56 00 01  44 b8 88 5e ce 37 00 00
0050  08 09 0a 0b 0c 0d 0e 0f  10 11 12 13 14 15 16 17
0060  18 19 1a 1b 1c 1d 1e 1f  20 21 22 23 24 25 26 27
0070  28 29 2a 2b 2c 2d 2e 2f  30 31 32 33 34 35 36 37
```

Figure 15 - Ping Packet UDP Transit Server -> Client

As observed above, at this stage the packet payload is not encrypted in either direction. Furthermore, there is no authentication of either the client or the server, therefore there is currently no guarantee of confidentiality.

7.1.2.1 Availability

At this stage we can guarantee availability, the client and server are fairly basic and therefore are not prone to complex error. To test this we allowed the client machine to ping the internal machine for an arbitrarily long amount of time in order to observe the number of packets dropped / lost and the latency the occurs.

```
--- 192.168.60.3 ping statistics ---
5544 packets transmitted, 5544 recei
ved, 0% packet loss, time 5627295ms
rtt min/avg/max/mdev = 0.504/1.228/3
7.392/1.087 ms
```

Figure 16 - UDP Availability Ping Test

As observed in Figure 16, the ping test was allowed to run for nearly an hour and a half, resulting in no packet loss with a maximum ms of 37, average ms of 1.2, which in regards to the criteria in *6.1.3* deems this test successful. One thing to note however is that if the server were to unexpectedly go down there would still be no acknowledgement of it client side, which is an issue of availability, further research is required in order to achieve this.

7.2 TCP/TLS

7.2.1 Implementation

This section of the virtual private network development cycle saw the shift of format and standard into that which was defined in 5.3 in order to accommodate for the additional functionality being added but also to maintain the previous. This occurred in both the client and server setups, functionality was moved to its relevant class and then included in the main through the header file. For example, the UDP functions used to be solely in main, but now the main contains a selection field to allow choice between UDP and TCP/TLS, the selection then directing to a separate function which serves as a "recipe" for functionality.

```
void udpRUN(){
  int tunfd, sockfd;
  tunfd = createTUNDevice();
  routeClient();
  sockfd = connectToUDPServer();

  //Enter main loop;
  udpProcess(tunfd, sockfd);
}
```

Figure 17 - New main.c UDP

Figure 17 is the most concise example of the organisation philosophy. The creation of a TUN device, and the the configuration of its IP and routes occur in both UDP and TCP/TLS, therefore createTUNDevice() and routeClient() both exist in the misc.c. Whereas connectToUDPServer() and udpProcess() are both unique to UDP, and are therefore present in udp.c. This design philosophy is reflected across the implementation where possible.

As previously mentioned, this section saw the implementation of TCP with TLS. Using TLS we allow the client and server to decide upon a mutual encryption methodology and authenticate each other using a Public Key Infrastructure method. In our particular implementation we utilised self-signed certificates which used sha256 with RSA encryption and a public key length of 1024.

Additionally, a few extra quality of life functions were implemented in order to make development and operation easier in the long run. Within tls.c one can find the function void processErr, which allows for the explicit detailing of what error may occur during the SSL context setup. Furthermore, in misc.c there is a function called int rootCheck, which does exactly what you'd expect, both the client and server require root to perform routing functions. RootCheck is always called first, and therefore does not allow either to run before being ran in root.

7.2.2 Testing

7.2.2.1 Confidentiality / Integrity

TLS over TCP was successfully implemented in this section, as observed in Figure 18 below, the TLS handshake takes place. Within this handshake the client authenticates the server using a Public Key Infrastructure exchange where the server sends its own certificate to the client, the client compares this with its own certificate authority key and either approves or denies the server.

Source	Destination	Protoco	Length	Info
10.0.2.4	10.0.2.5	TCP	66	33440 → 4433 [ACK] Seq=
10.0.2.4	10.0.2.5	TCP	371	33440 → 4433 [PSH, ACK]
10.0.2.5	10.0.2.4	TCP	66	4433 → 33440 [ACK] Seq=
10.0.2.5	10.0.2.4	TCP	1074	4433 → 33440 [PSH, ACK]
10.0.2.4	10.0.2.5	TCP	66	33440 → 4433 [ACK] Seq=
10.0.2.4	10.0.2.5	TCP	256	33440 → 4433 [PSH, ACK]
10.0.2.5	10.0.2.4	TCP	292	4433 → 33440 [PSH, ACK]
10.0.2.4	10.0.2.5	TCP	143	33440 → 4433 [PSH, ACK]
10.0.2.5	10.0.2.4	TCP	220	4433 → 33440 [PSH, ACK]
10.0.2.4	10.0.2.5	TCP	66	33440 → 4433 [ACK] Seq=

Figure 18 - TLS Handshake

After the server authentication, the key part of this handshake is the establishing of a common encryption method for the client and server to use in order to privatise communication between themselves, this is also successfully implemented, we have for the means of testing allowed the VPN to print its authentication process and what encryption method it has established. Figure 19 shows the certificate verification callback as well as the determined encryption method as printed in the command line client side.

```
subject= /C=UK/ST=Gloucestershire/O=Ja
cob/OU=Jacob/CN=jwilliams.com/emailAdd
ress=holowi3731@gotkmail.com
Verification passed.
SSL connection is successful
SSL connection using AES256-GCM-SHA384
```

Figure 19 - Successful Authentication and Encryption Established

Using the authentication and encryption allows us to then utilise TCP connectivity to transmit our packets into the VPN. Using TCP gives us the additional benefits of being more confidential thanks to encryption and also guaranteeing integrity thanks to TCP's functionality of acknowledging when packets are received and innate error detection.

In Figure 20 we can see the progress of the packet client side, the ping packet generated within the TUN interface, which is then stored in the payload of a TCP packet and

encrypted. In Figure 21 we see the packet progress server-side, and then in Figure 22 we see the comparison of the raw ping packet to the encrypted TCP payload.

Source	Destination	Protocol	Length	Info
10.4.2.99	192.168.60.3	ICMP	100	Echo (ping) request id=0x1799,
10.0.2.4	10.0.2.5	TCP	181	33674 → 4433 [PSH, ACK] Seq=385
10.0.2.5	10.0.2.4	TCP	181	4433 → 33674 [PSH, ACK] Seq=329
10.0.2.4	10.0.2.5	TCP	68	33674 → 4433 [ACK] Seq=38564817
192.168.60.3	10.4.2.99	ICMP	100	Echo (ping) reply id=0x1799,

Figure 20 - Ping packet TCP Client-side

Source	Destination	Protoco	Length	Info
10.0.2.4	10.0.2.5	TCP	181	33716 → 4433 [PSH, ACK]
10.4.2.99	192.168.60.3	ICMP	100	Echo (ping) request id=
10.4.2.99	192.168.60.3	ICMP	100	Echo (ping) request id=
192.168.60.3	10.4.2.99	ICMP	100	Echo (ping) reply id=
192.168.60.3	10.4.2.99	ICMP	100	Echo (ping) reply id=
10.0.2.5	10.0.2.4	TCP	181	4433 → 33716 [PSH, ACK]
10.0.2.4	10.0.2.5	TCP	68	33716 → 4433 [ACK] Seq=

Figure 21 - Ping Packet TCP Server-side

```
0000  00 04 ff fe 00 00 00 00  00 00 00 00 00 00 08 00   ........ ........
0010  45 00 00 54 8c b0 40 00  40 01 a4 e6 0a 04 02 63   E..T..@. @......c
0020  c0 a8 3c 03 08 00 7e 2f  19 12 00 ae 42 37 8f 5e   ..<...~/ ....B7.^
0030  9c 77 07 00 08 09 0a 0b  0c 0d 0e 0f 10 11 12 13   .w...... ........
0040  14 15 16 17 18 19 1a 1b  1c 1d 1e 1f 20 21 22 23   ........ .... !"#
0050  24 25 26 27 28 29 2a 2b  2c 2d 2e 2f 30 31 32 33   $%&'()*+ ,-./0123
0060  34 35 36 37                                       4567

0000  00 04 00 01 00 06 08 00  27 60 51 63 00 00 08 00   ........ '`Qc....
0010  45 00 00 a5 00 26 40 00  40 06 22 25 0a 00 02 04   E....&@. @."%....
0020  0a 00 02 05 83 b4 11 51  14 9d 3f 3b 04 8b 8e e2   .......Q ..?;....
0030  80 18 01 1b 18 a0 00 00  01 01 08 0a 00 11 bd 79   ........ .......y
0040  00 11 bb 92 17 03 03 00  6c 3f f1 7f d0 30 b5 95   ........ l?...0..
0050  35 06 f2 6b 1f 28 59 be  bc a5 eb 78 1e f6 db b3   5..k.(Y. ...x....
0060  3d e0 ed 1d ab 6e f1 74  45 3d fa 2f c8 ee d0 af   =....n.t E=./....
0070  9b 7c 43 b4 92 3f c6 ad  1d 6f a6 cc 44 9b df 74   .|C..?.. .o..D..t
0080  21 fa ee c5 20 3f d2 dd  96 c0 49 c9 7e 76 f5 20   !... ?.. ..I.~v.
0090  5a 33 96 8b 52 2e f4 f7  50 1b 27 ca 3d 28 a2 8f   Z3..R... P.'.=(..
00a0  e0 b9 24 49 25 5f 8d 48  c0 11 f7 ec b7 ec ae 9a   ..$I%_.H ........
00b0  74 f3 8b c2 7b                                    t...{
```

Figure 22 - Raw Ping Packet (TOP) Vs TCP Payload (Bottom)

Source	Destination	Protoco	Length	Info
10.4.2.99	192.168.60.3	ICMP	100	Echo (ping) request
192.168.60.3	10.4.2.99	ICMP	100	Echo (ping) reply

Figure 23 - Internal TCP Ping

As aforementioned in 6.2.1, there is no server-side authentication of the client at this point, this will be explored in the next sprint. Despite this, everything tested for in regards to confidentiality and integrity in this sprint has passed.

7.2.2.2 Availability

The uptime of the implementation is reliable in regards to once a connection has been established it is reliable, below in Figure 23 you can see the results of up to an hour of constant pinging.

```
--- 192.168.60.3 ping statistics ---
3515 packets transmitted, 3515 receiv
ed, 0% packet loss, time 3546015ms
rtt min/avg/max/mdev = 0.574/1.666/18
.734/1.200 ms
```

Figure 24 - TCP/TLS Availability Pinging

The average and minimum latency is similar to that of the UDP test, but the maximum latency is over 20 ms less than that of UDP. Similarly, TCP experienced zero packet loss. In regards to disconnection and reconnection, both the client and the server failed the tests, peculiarly once the connection had been terminated on one side the other side would appear to experience a flood of packets through the tunnel despite such an event not being detectable through WireShark. Re-connectivity functionality will be explored in a future sprint, or may even be organically ironed out in the planned Multi-client functionality sprint.

7.3 Client Authentication

7.3.1 Implementation

In order to retain security during the exchange and confirmation of username and password, the login process is conducted after the establishment of the TCP/TLS session. In this way we can assure confidentiality and integrity so long as the TCP/TLS functionality is still utilised. To achieve client authentication the method adopted was to use the servers shadow file in order to compare a sent and received username and password against the one on the server machine.

7.3.2 Testing

7.3.2.1 Confidentiality and Integrity

The overall confidentiality and integrity of the communication using TCP/TLS was tested in 7.2.2.1, and has not been modified and therefore will not be retested here. Overall, the username and password verification implementation was a success, and the client now transmits the input of username and password to the server in packets encoded using TLS. After processing the inputs against the shadowfile, the server returns either match or mismatch, and then both the server and client proceed as you'd expect depending on the result. The implementation responds correctly to all permutations of possible entry, these being: correct username, correct password (Figure 25); correct username, incorrect password (Figure 26); incorrect username, correct password (Figure 27); and incorrect username, incorrect password (Figure 28).

All of the below figures are formatted with the client input on top, and the server output below.

```
Verification passed.
SSL connection is successful
SSL connection using AES256-GCM-SHA384
Enter username:
seed
Enter password:
dees
Server returns: Authentication Successful, Connection Established.
SSL connection established!
Given Username: seed
Given password: dees
Client Authentication Successful, establishing connection.
```

Figure 25 - Correct Username, Correct Password

```
Enter username:
seed
Enter password:
wrong
Auth Failed. Terminating.
Given Username: seed
Given password: wrong
No Match.
Auth failed, terminating child process.
```

Figure 26 - Correct Username, Wrong Password

```
Enter username:
wrong
Enter password:
dees
Auth Failed. Terminating.
No password for username
Auth failed, terminating child process.
```

Figure 27 - Incorrect username, Correct password

```
Enter username:
wrong
Enter password:
wrong
Auth Failed. Terminating.
No password for username
Auth failed, terminating child process.
```

Figure 28 - Incorrect Username, incorrect password

All of the printed statements showing the username and password exist only for testing, and will be removed for general usage and demonstration. The only key security flaw recognised in this implementation is the fact the password input on the client side is not hidden, and therefore is susceptible to “over the shoulder” theft. This issue most certainly be further addressed in the later QoL sprints.

7.3.2.2 Availability

In regards to availability, the login function occurs before general operations, and therefore does not modify it in anyway. Given this, the results of the availability testing for TCP/TLS in section 7.2.2.2 still stand. The client and server can still maintain steady connection with negligible loss and fairly fast speeds. The issues identified in 7.2.2.2 still stand but now in a further manner. Given the event of an incorrect login, both the client and server will need to be restarted to allow for reconnection. The client does seem to be able to reconnect to the server, but then after a second experiences a flood of packets from the tunnel that do not appear in WireShark. The plan remains to iron this out in the next sprint in order to make way for multi-client functionality, since the two will need to go hand-in-hand.

7.4 Multi-client Functionality

7.4.1 Implementation

The first step of this stage was to solve the re-connectivity issue that had been plaguing the implementation in previous stages, the thought being that if re-connecting wasn’t possible then it may interfere with multi-connectivity, both in whether more than one could connect and in if one disconnects would it affect the other. The issue was determined to be the nature of how the client and server work after a connection has been terminated. The key lies in with how we close the client, since it uses blocking functions to operate, there is no way to manually tell it to close and therefore we must resort to using a signal interrupt to close it.

Realising this was the key to solving the problem, since when the client was interrupted, it would close and free all constructs it had created, but this would not be communicated to the server which would keep its constructs open. Therefore we implemented a method to catch the signal interrupt, and have it direct to a method called closeHandler() which would send a final message to the server informing it that the client connected on that child has been terminated. Then on the server side, we have to implement the function clientclosed() which checks the packets coming through the socket for a termination signal, which if recognised will then perform its own termination handling. At this stage the certificates were also updates, mainly as to not contain any personally recognisable details but also to update the key to a stronger state, this time being AES256 with a key length of 2048 bits.

The multi-client functionality was already partially, implemented. Every time a connection was received the program would fork into two processes, the parent that would loop back around to handle any other connections and the child which would enter the if statement only it can enter. Creating a multi-client system was merely a case of modifying this existing structure to allow multiple connections and direct traffic to the separate child processes. This was done using the pipe() system, which allowed the establishing of a bridge between the parent process and its child processes by creating a structure akin to the tunnel but within the process itself.

7.4.2 Testing

7.4.2.1 Confidentiality and Integrity

This section is built off of the back of the previously established TCP/TLS protocol functionality, and therefore each now connection established will still undergo perform and undergo authentication and have a SSL context established. Figure 29 shows the connection of two clients each with their own shown SSL context.

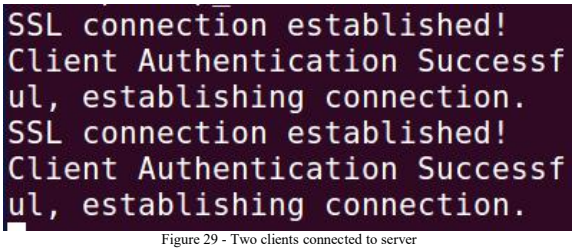

Figure 29 - Two clients connected to server

7.4.2.2 Availability

Unfortunately availability is where this stage of the implementation falls apart. The connectivity is flawless but when it comes to long periods of usage by both clients simultaneously there is much packet loss. The server seems to work reliably when two clients are connected but only one is working over a long period of time, and similarly availability is still assured when only using one client.

7.5 Quality of Life Functionality

This section will details various quality of life / quality of service functions and changes made throughout development and further in more detail than what was referenced in the other sections.

7.5.1 Client Closed Notifications

When a client is disconnected from the TCP/TLS session, the server needs to know so it can close its relevant structures and free resources. However, the only way to terminate the client is to perform an interrupt (control + c), therefore we needed to implement a catch for signal(SIGINT) that directs the program to a closehandler when performed. Using signal.h allows us to define the behaviour that occurs when a signal interrupt is performed, in the case of this implementation it redirects the program to the closehandler, which is shown in Figure 30.

```c
void closeHandler(){
  //Initialise closing.
  printf("\nClosing...\n");
  char buff[BUFF_SIZE] = "&client&closed";
  buff[strlen(buff)] = '\0';

  //Tell server client has closed, close all relevant structs.
  if(ssl != NULL){
    SSL_write(ssl, buff, BUFF_SIZE-1);
    SSL_shutdown(ssl);
    SSL_free(ssl);
  }
  close(TLSsockfd);
  close(TLStunfd);
  exit(0);
}
```

Figure 30 - Clientside closeHandler()

This writes to the server that the client has closed, and then performs a safe shut-down of all its currently used resources for good measure. On the server side we implement clientClosed(), which sits nested in the TLSsocketSelected function, and checks the packets for the unique close identifier "%client%closed". If it finds it, then it enters the IF statement it is checked against and performs a safe shut-down of all the SSL structures currently running on the child process, as shown in Figure 31.

```c
void clientClosed(char buff[], SSL* ssl, int sockfd){
  //Only does anything if this IF statement is true.
  if((buff[0] != '\0') && strstr(buff, "&client&closed") != NULL){
    printf("Client has been closed, terminating connection...");
    if(ssl != NULL){
      SSL_shutdown(ssl);
      SSL_free(ssl);
    }
    close(sockfd);
    exit(0);
  }
}
```

Figure 31- Server side clientclosed()

Figure 32 shows how the server side looks when a client is connected, interrupted client side, then reconnected.

```
SSL connection established!
Client Authentication Successful, establishing co
nnection.
Client has been closed, terminating connection...
SSL connection established!
Client Authentication Successful, establishing co
nnection.
```

Figure 32 - Close handler demo

7.5.2 rootCheck

The simplest implementation, both the client and server require root access in order to operate. Therefore, in order to create zero confusion of this fact, rootCheck was implemented to detect, and the user, of the root privileges if they are needed.

```c
int rootCheck(){
  if(getuid() != 0){
    printf("This server requires root access to function correctly.\n");
    abort();
  } else {
    return 0;
  }
}
```

Figure 33 - rootCheck

```
./main jwilliams.com 4433
This server requires root access to function correctly.
Makefile:5: recipe for target 'run' failed
```

Figure 34 - rootCheck demo

7.5.3 Removal of Hard-coded Addresses

More of a structural change, the client was modified within both UDP and TCP/TLS to not utilise hardcoded addresses. This promotes not only interoperability, but also encourages better security. Without hardcoded variables, if a malicious third party was able to get a hold of a version of this implementation, they would merely have the means to connect to a server they know, and wouldn't be able to discern who the VPN client belonged to before.

7.5.4 SSL Explicit Error Detailing

In order to perform better debugging, a method of receiving more information on SSL errors was researched and implemented in the form of identifying key errors likely to affect the program. This error is calculated when the SSL connection is established, but at its simplest it merely outputs a number, using processErr() we can get the full name of the error for further inquiry.

```c
void processErr(int err, SSL* ssl){
  //Get error code
  int errorCode = SSL_get_error(ssl, err);

  //Diagnoses
  switch(errorCode){
    case SSL_ERROR_NONE:
      break;
    case SSL_ERROR_ZERO_RETURN:
      fprintf(stderr,"SSL connect returned 0.\n");
      break;
    case SSL_ERROR_WANT_READ:
      fprintf(stderr,"SSL connect: Read Error.\n");
      break;
    case SSL_ERROR_WANT_WRITE:
      fprintf(stderr,"SSL connect: Write Error\n");
      break;
    case SSL_ERROR_WANT_CONNECT:
      fprintf(stderr,"SSL connect: Error  connect.\n");
      break;
    case SSL_ERROR_WANT_ACCEPT:
      fprintf(stderr,"SSL connect: Error accept.\n");
      break;
    case SSL_ERROR_WANT_X509_LOOKUP:
      fprintf(stderr,"SSL connect error: X509 lookup.\n");
      break;
    case SSL_ERROR_SYSCALL:
      fprintf(stderr,"SSL connect: Error in system call.\n");
      break;
    case SSL_ERROR_SSL:
      fprintf(stderr,"SSL connect: Protocol Error.\n");
      break;
    default: fprintf(stderr,"Failed SSL connect.\n");
  }
  CHK_SSL(err);
}
```

Figure 35 - processErr()

8.0 Evaluation

The development of the Virtual Private Network implementation has revealed much about the systems which were previously a relative mystery. Before the only interaction with a Virtual Private Network had was using one that allows you to direct your traffic through another country, and it wasn't known that a VPN can be used in such a way that allows you to connect to an internal network from the outside as such. Developing a VPN has taught much about how to utilise various network programming methods, as well as the utilisation of TLS and certificates, which I hadn't done before.

The one personal drawback of this project was the fact that there wasn't enough time to explore absolutely the current state of Virtual Private Networks. It would have been good to explore the concept of permanently linking two networks via a Gateway-To-Gateway VPN. It would also have been interesting to test the VPN implementation with more than two connections, but due to the limitations of hardware this wasn't possible. Similarly, it is felt that while this delve

into VPNs was beneficial, it was only exploring the tip of the iceberg. Whilst performing the research for the Literature Review section, it was fairly evident that there are many advanced implementations as well as up-and-coming methods of implementing VPNs which contained terminology never seen before.

In the future, we would like to research the full time line and progression of VPNs, this way we can understand the contemporary research that we currently do not understand. In terms of the implementation, further development would be to create a server that is able to reliably handle numerous clients, and creating a new functionality within both client and server that allows them to act as a Gateway-to-Gateway VPN.

9.0 Conclusion

The Virtual Private Network was once a technology that was considered expensive and complicated to implement into the everyday work flow, but now it is possible to create a rudimentary version with relative ease. Furthermore, more advanced implementations are cheaper than ever, and whilst they certainly will have an upfront cost for purchase and implementation time, in the long run it will be beneficial to your business. The problems there in lie that with the increased popularity comes an increased potential to misunderstand the concept of the VPN and believe that simply having one will be all that is required to improve work flow and security. This isn't the case, in fact it is debatable whether Virtual Private Networks themselves actually improve security, whilst it certainly creates a secure way to access an internal network from the outside, opening up such an avenue can create many additional problems. Therefore if one is looking to implement a VPN, one must consider if it is really necessary, and if so, who has access to it; limiting who can access the VPN is one of the best ways to reduce potential of danger created by the new access method. In finality, the VPN business is only getting more and more popular, and whilst commercial VPNs for home users are security oriented, one has to question the security characteristics of a VPN that connects networks. But this will all be addressed eventually, as with the business itself the amount of research surrounding the adaptivity and implementation of them is only growing.

10.0 References

Alshalan, A.A., Pisharody, S.P. and Huang, D.H. (2016) A Survey of Mobile Vpn Technologies. Ieee Communications Surveys and Tutorials [online]. 18 (2), pp. 1177-1196. [Accessed 02 February 2020].

Caldwell, T.C. (2012) Locking Down the Vpn. Network Security [online]. 2012 (6), pp. 14-18. [Accessed 05 February 2020].

Carmouche, J.H.C. (2006) Ipsec Virtual Private Network Fundamentals. Usa: Cisco Press.

Dimension Data (2014), Secure Mobility Survey Report[Online]. Available from: http://www.dimensiondata.com/Global/Downloadable Doc-uments/Secure Mobility Survey Findings Report.pdf

Longsworth, J.L (2018) Vpn: From an Obscure Network to a Widespread Solution. Computer Fraud and Security [online]., pp. 14-15. [Accessed 02 February 2020].

Khanvilkar, S.K. and Khokhar, A.K. (2004) Virtual Private Networks: An Overview with Performance Evaluation. Ieee Communications Magazine [online]. 42 (10), pp. 146-154. [Accessed 04 February 2020].

Raghavan, N.R., Gopal, R.G., Annaluru, S.A. and Kura, S.K. (2002) Virtual Private Networks and Their Role in E-business. Bell Labs and Technical Journal [online]. 6 (2), pp. 99-115.

Rossberg, M.R. and Schaefer, G.S. (2011) A Survey on Automatic Configuration of Virtual Private Networks. Computer Networks [online]. 55, pp. 1684-1699. [Accessed 04 February 2020].

Wenliang, W.D. (2017) Computer Security: A Hands-on Approach. 2nd ed. : Createspace Independent Publishing Platform.