University of Hertfordshire
School of Computer Science

Modular BSc Computer Science (Networks)

6COM1053 – Computer Science Project

Final Report
April 2021

How to solve some Internet Privacy and Security Issues and Real-world Security Threats

Name: Anthony Constant

Supervised by: Chris Treglohan

Abstract – Executive Summary

The main purpose of this final project report was to implement our own Virtual Private Network(VPN) server, with the purpose of overcoming some Internet Privacy and Security issues. Also, to implement a motion detection sensor, with the purpose to overcome potential real-world security threats. The project consisted of several different tasks, which were aimed at overcoming these issues. To prepare it was necessary to follow specific steps to creating a VPN server using WireGaurd protocols. Additionally, building a motion detection sensor using a PIR motion sensor, Arduino Uno Micro-controller board, Java programming language in Arduino IDE and LED bulb.

The results of creating the VPN server exposed many advantages that were further investigated. Some of the advantages included the algorithm, encryption, and ports. The results of implementing the VPN server was successful however, there were some issues encountered.

The results of building the motion detection sensor revealed many advantages, which could be potentially implemented. Some of the advantages consisted of adding extra code to control the behaviour of the PIR motion detection sensor when it detects motion. The results of implementing a motion detection sensor was successful however, there were many issues encountered throughout building it.

The conclusions that could be drawn from this final project report were that not all the attempts had worked. Such that, it was necessary to refer to the drawing board and think of a new design and implementation. Overall, after several attempts on the implementation and testing were successful. However, it was essential to gain a better understanding of the topic to overcome these failed attempts and overall acquired better insights into the topic, by ultimately building upon previous designs to overcome the issues that had arisen.

Table of Contents

1.0 Introduction

This final project report had been completed in response to the brief, for determining and evaluating the most suitable Virtual Private Network(VPN) protocols to implement which could overcome Internet privacy and security issues as well as building a PIR motion detection sensor, to detect potential real world threats.

The first part of the project involved the research for it, in the form of Internet research which were analysed in detail. The second part consisted of conducting of gathering useful resources to create the requirements and specifications. The next part consisted of the design and implementation such that, it was also essential to conduct tests on these artefacts.

This final project report describes the work that was performed during all the phases mentioned, then will be conducting an overall discussion and evaluation. Ultimately, the final project report is intended to overcome at least some Internet Privacy and Security issues as well as detecting potential real-world threats. The intended purpose of building this sort of set-up is to protect home devices and to gain a deep understanding of the following topics: Networking, Cyber-Security, Programming and working with hardware components.

Before delving deeper into the topic, it is essential to understand the definitions of privacy and security in cyber security, to ensure the brief is overcome.

- Data Privacy – In general, this refers to how the data is handled from a VPN point of view. For example, if a user is running ExpressVPN, how is their data being handled? i.e. is their data being stored or shared?
- Data Security - Essentially, security is a long chain of processes and tools and if the chain is broken, then ultimately the security has been ‘breached’. Overall, the security chain is only as strong as the weakest link.

2.0 Literature Review and Research

In the next part of the report it will be discussed the research and review undertaken to create a Virtual Private Network Server. The next part is the research on how to build a motion detection sensor, then we'll be discussing the requirements and specifications required to undertake the project.

2.1 What is a Virtual Private Network(VPN)?

A Virtual Private Network(VPN) is essentially an encrypted connection from an end device to a network. The encrypted connection enforces sensitive data to be safely transmitted, without any interferences. There are two different VPN access types for example, remote access which can be described as computer to network connection, connecting users to the private network from various locations. Site-to-site which can be described as network to network, connecting multiple sites over a public network. *(Cisco, 2021)* However, the type of access which has been implemented within this project report, will be remote access.

2.2 What is Encryption and Decryption?

Encryption is a method of transforming readable data or plain text into a form that is unreadable which is called cipher text. This enables transmission of data into a form that is unreadable, and which remains confidential and private. Decryption is a method to transform cipher text back into readable plain text.

OpenVPN and WireGaurd use encryption algorithms to encrypt the data, such as: Data Encryption Standard(DES), Advanced Encryption Standard(AES), Rivest-Shamir-Adleman(RSA). AES and DES implement symmetric key systems and has only ONE key such that, it requires a secret key i.e. a password to decrypt the file. Whereas RSA implements Asymmetric key systems, which consists of TWO keys, a public and private key. Such that if you encrypt with the private key you need the public key to decrypt and vice-versa. *(Ajay Ohri, 2021)*

	Advantages	Disadvantages
Symmetric (one key)	• Fast • Strong	• Uses same key for encryption and decryption
Asymmetric (public & private key)	• Better key distribution • Scalability	• Slow • Mathematically Intensive

Cryptographic technology provides for the following:

- Confidentiality
- Authentication
- Nonrepudiation
- Integrity

In the next part, OpenVPN and WireGaurd will be discussed in further detail, in which could potentially be used to overcome some Internet privacy and security issues.

2.3 OpenVPN

OpenVPN is free open-source VPN software and was first built and released in late 2001. *(OpenVPN, 2021)* By default, it uses the Blowfish algorithm designed by Bruce Schneier in 1993, which is an encryption technique identical to the Data Encryption Standard(DES). *(AbhayBhat, 2019)*

Furthermore, OpenVPN could be configured to either UDP or TCP protocol, which are both transport layer protocols within the OSI model and used to transmit data over the network. However, the manufactures documentation suggests that "OpenVPN works best over a UDP connection." *(OpenVPN, 2021)*

Overall, OpenVPN has been recommended by many security experts and suggests it's the safest VPN protocol to implement and use. It could also be said that, OpenVPN is a highly secure protocol, which uses 256-bit symmetric key encryption system and consists of Secure Socket Layer (SSL) protocols to authenticate and encrypt communication over the network.

2.4 WireGaurd

WireGaurd is free open-source VPN software and was officially released and announced March 29th, 2020. By default, WireGaurd uses various encryption techniques such as: ChaCha20 for symmetric key encryption with Poly1305 for message authentication, a combination that is proven more performant than AES in this instance. *(Sven Taylor, 2020)*

Overall, WireGaurd has also been recommended by many security experts and is more recent in VPN technologies. WireGaurd only supports a UDP connection, which is best suited in both instances, as UDP provides for significantly greater speeds than TCP.

2.5 OpenVPN Vs WireGaurd

It is now essential to decide whether to choose OpenVPN or WireGaurd, to implement as the server to overcome some Internet privacy and security issues. It has been decided to choose WireGaurd for the following reasons:

- Security/Encryption – Overall, WireGaurd provides for better encryption over OpenVPN. It could be said that OpenVPN uses OpenSSL encryption and supports a variety of different algorithms including AES, Blowfish, Camellia, ChaCha20 and more for encryption and authentication. It also uses, MD5, MD4, SHA-1, SHA-2, BLAKE2, and more for hashing. However, the range of algorithms means OpenVPN is more agile and as a result, makes the code more complex, which ultimately leads to slower encryption. WireGaurd is unique and with each version implements a fixed number of algorithms such as ChaCha20 for symmetric encryption, Poly1305 for authentication, Curve25519 for Elliptic-curve Diffie-Hellman (ECDH) key agreement and BLAKE2s for hashing. Furthermore, OpenVPN implements certificates for identification and encryption whereas, WireGaurd uses public key encryption for certain tasks. *(Heinrich Long, 2021)* Overall, because OpenVPN is more flexible in terms of protocols, which leads to more complexity, WireGaurd is ultimately better for encryption. As WireGaurd only includes one set of protocols and ciphers, which means less code complexity.

- Speed – WireGaurd protocol runs much faster than OpenVPN. It was found from research that, under a variety of circumstances using a speed tester, WireGaurd outperformed OpenVPN in all instances. It could be said that, WireGaurd is about 58% faster than OpenVPN across different locations, using the results from testing. *(Heinrich Long, 2021)*

- Privacy – VPN protocols provide security but doesn't necessarily mean they provide privacy. Privacy relies dependently on whether the policies of the VPN services are required to store logs, which ultimately defines the user's privacy. However, WireGaurd was designed for mainly speed and security. The problem with WireGaurd privacy is that as part of the encryption algorithm, it connects the public keys to the authorised IP addresses and as a result, stores the IP addresses on the VPN server for long periods of time or until rebooted. Compared to OpenVPN, it does not store any IP addresses on the server, which means ultimately OpenVPN would be preferred in terms of privacy. However, it is possible to come up with different solutions, to overcome the WireGaurd privacy issue such that, NordVPN developed a similar implementation to overcome this issue. *(Heinrich Long, 2021)*

  As a result, it was decided to implement a WireGaurd VPN server, to overcome some Internet privacy and security issues. The next part will decide how to implement a VPN server.

	Advantages	Disadvantages
OpenVPN	• Privacy	• Slow • Less Security
WireGaurd	• Fast • Better Security	• Privacy

2.6 Implementing a Virtual Private Network(VPN) Server

As part of the implementing the WireGaurd VPN server, a Raspberry Pi 4 was deployed, with PiVPN installed. After gathering research, it was recommended to run a WireGaurd server on a Linux system, to prevent compatibility issues. It was later discovered from further research, that PiVPN would be best suited to set-up and run a WireGaurd server. *(Emmet, 2020)* The main chapters of the project report will expose exactly how the server was implemented and able to have users connect to counter internet privacy and security issues.

2.7 What is a PIR motion detection sensor?

Passive infra-red(PIR) are essentially used in security systems such as triggering an alarm to alert an administrator. The PIR will act as the motion detection sensor by the sensors inside them, to detect small changes in the infrared waves which happens when something passes within the sensors view.

For example, if a human walk past the sensors range of view it, will essentially detect small changes between the background temperature, against the human's body temperature and is highly sensitive. Furthermore, the PIR device can convert the radiation detected from its sensors into a voltage output, sounding the alarm. *(Laura, 2019)*

2.8 How to implement a PIR motion detection sensor

As part of implementing the PIR motion detection sensor, initially a Raspberry pi 4 and PIR sensor was deployed with the additional hardware**: Breadboard, Jumper cables, Resistors, LED bulbs**. Additionally, python code was used to develop the code, to control the behaviour of the PIR sensor when it detected motion, within different IDE's. *(Tech With Tim, 2019)* There were several failed attempts before developing the full working artefact however, it could be said it was easier to build and continue developing upon each failure to gain a deeper understanding, which will expose how the PIR motion detection sensor was eventually built, in order to detect potential real-world threats, within the main chapters of the project report.

3.0 Chapter 1: How to implement and maintain a WireGaurd Virtual Private Network(VPN) server

In this part of the report it will be discussed the first phase of setting up and running a WireGaurd VPN server, to overcome some Internet privacy and security issues. The second phase will consist of building a PIR motion detection sensor to detect potential real-world threats.

3.1 Getting started how to implement and host a WireGaurd VPN Server

As part of implementing and hosting a WireGaurd VPN server, it will consist of gathering the following resources, as discussed previously in the report. However, the next part, will expose the exact list of prerequisites to create it.

3.2 Prerequisites

Name	Component	Description
Hardware	Raspberry Pi 4	To use the OS terminal to run and install PiVPN / WireGaurd server
	SD Card (>=8GB)	To be the storage of the Raspberry Pi. Raspberry OS installed onto the SD card. The official Raspberry Pi image file: https://www.raspberrypi.org/software/ (RaspberryPi, 2021)
	HDMI monitor, Keyboard, Mouse	To be the Raspberry Pi's peripherals
Software	PiVPN	To set-up and install WireGaurd VPN server
	WireGaurd client	To establish the connection between the server to any end device.
	No-IP	To create hostname for setting up the Static IP Address

3.3 How to implement and host a WireGaurd VPN server

3.3.1 Installing PiVPN on the Raspberry Pi 4

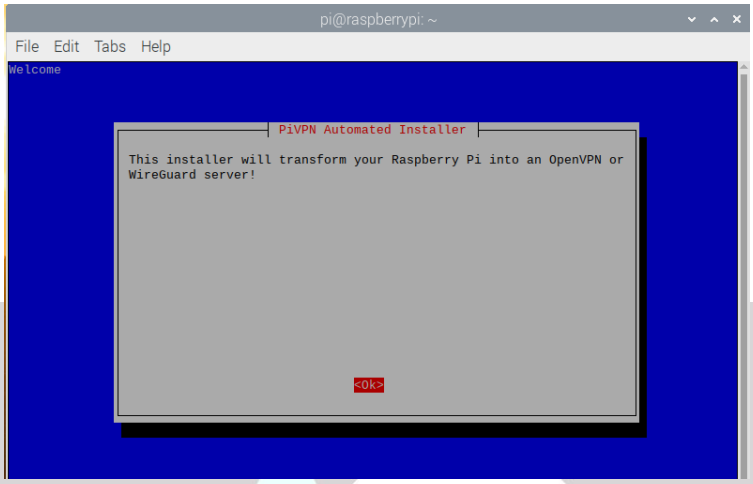
As part of the first phase, it was required to access the official PiVPN website to acquire the curl installation, to use within the Raspberry Pi 4 OS terminal. Furthermore, it was recommended to run the following commands before getting started.

- Sudo apt-get update – updates the list of available packages
- Sudo apt-get upgrade – installs latest versions of acquired packages

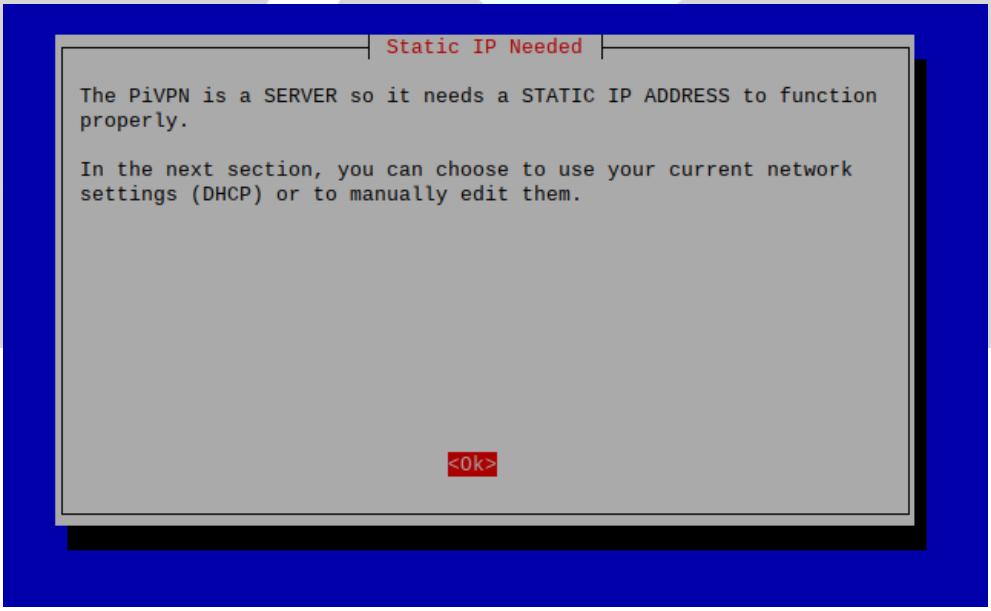
After copying the installation text as shown in the image above, it was then pasted into the OS terminal to run the BASH script. It was recommended, to run BASH scripts, from trusted sources.

3.3.2 WireGaurd server installation

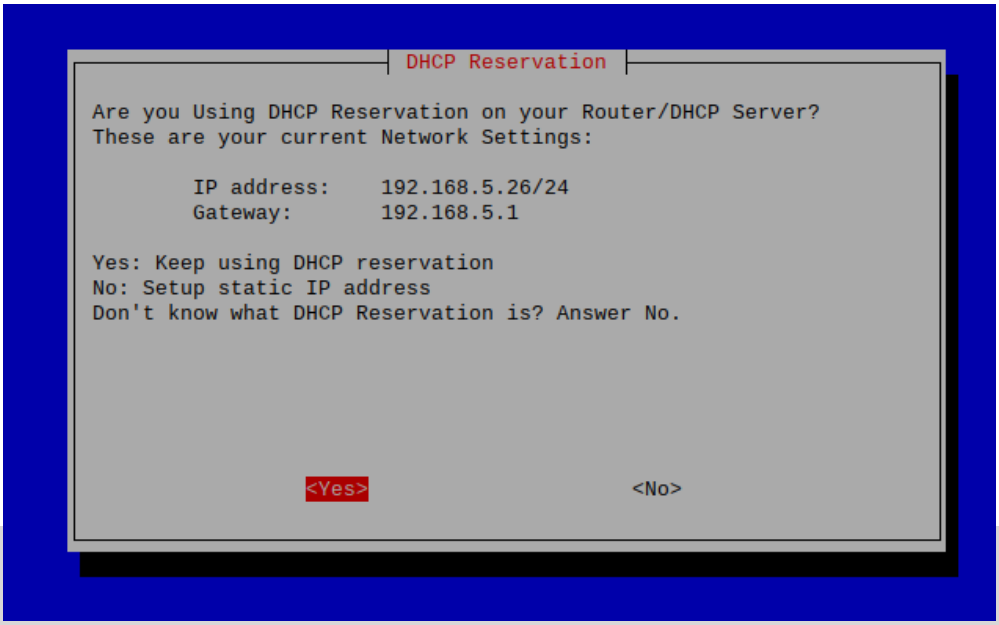
After installing PiVPN as shown in the image below. PiVPN will be used as guidance using the GUI provided, to ultimately set-up the WireGaurd server.



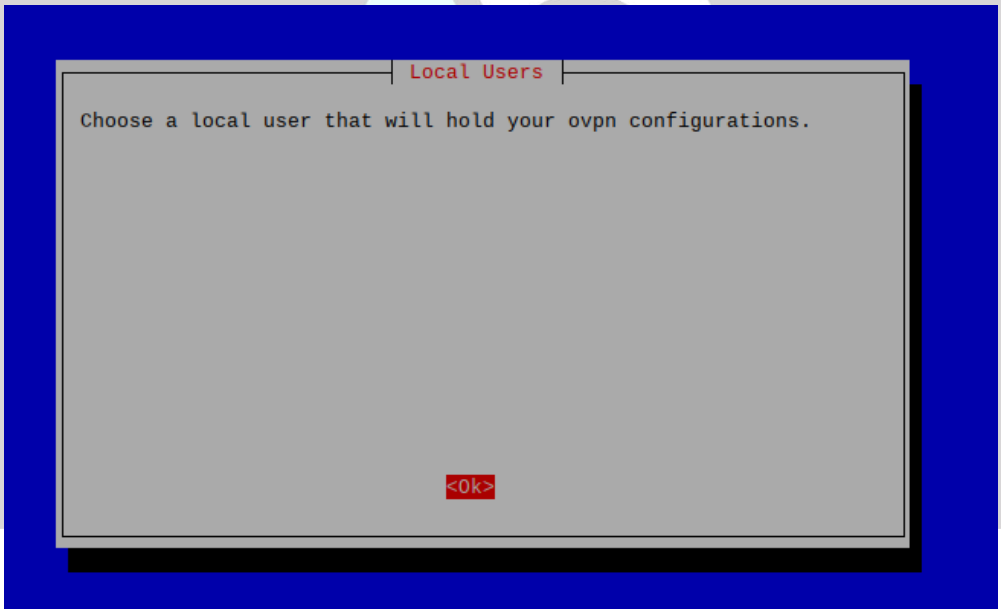
Next, the screen appeared as shown in the image below. PiVPN suggested to acquire a Static IP address. A Static IP address provides for an immutable internet address, like a street address in the real-world.



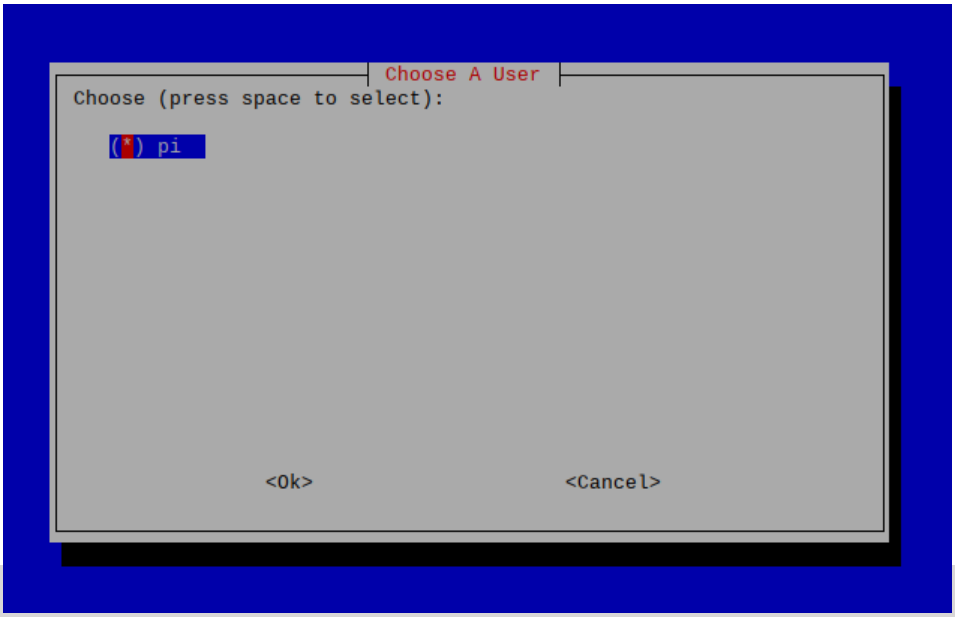
As you can see from the image PiVPN suggests having a static IP address set-up before continuing. A Static IP address will be set-up as shown in later steps, and therefore proceeded. Furthermore, PiVPN automatically detected the routers default gateway and assigned an IP address 192.168.5.26.



PiVPN required to choose a local user to store all permissions and therefore, picked Pi as the root user, as shown in the images below.

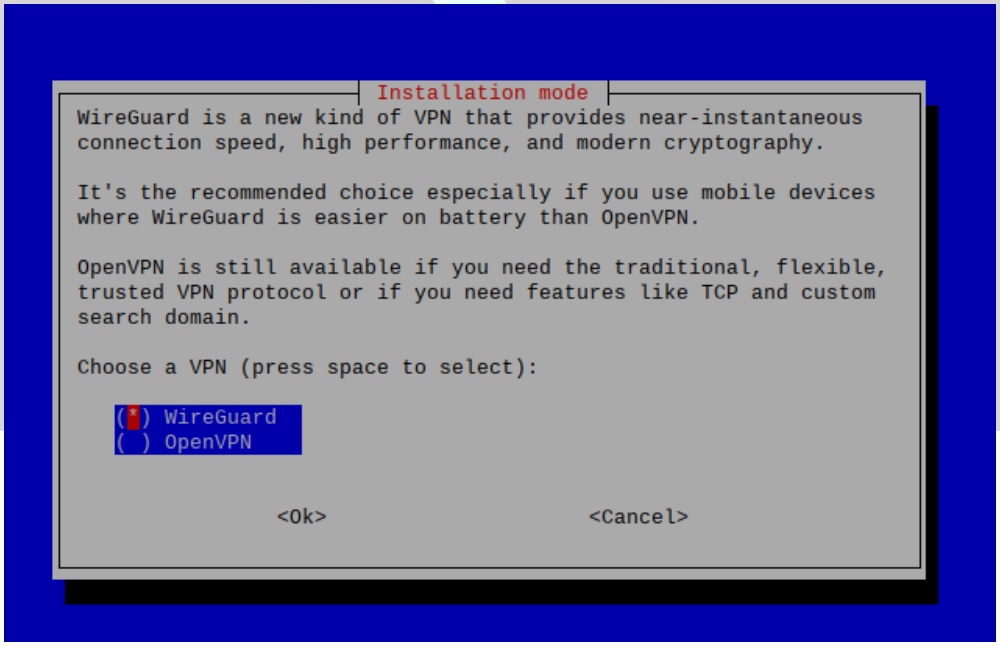


The image reveals choosing the host user for the VPN server.

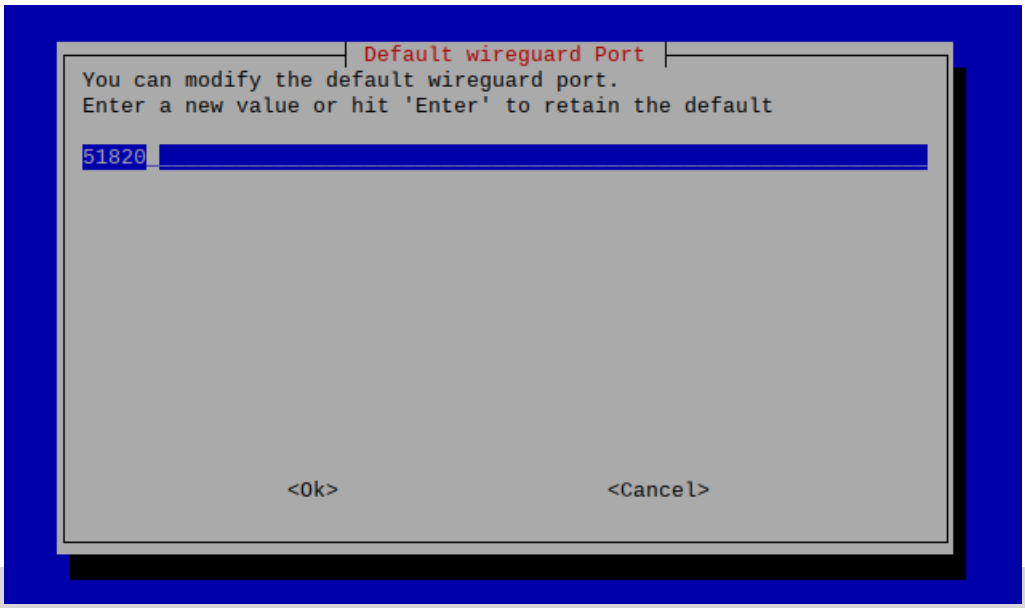


The image reveals choosing “pi” as the host for the VPN server.

PiVPN required selecting between either WireGaurd or OpenVPN, as discussed previously in the report, it was decided to choose WireGaurd due to the overall advantages.

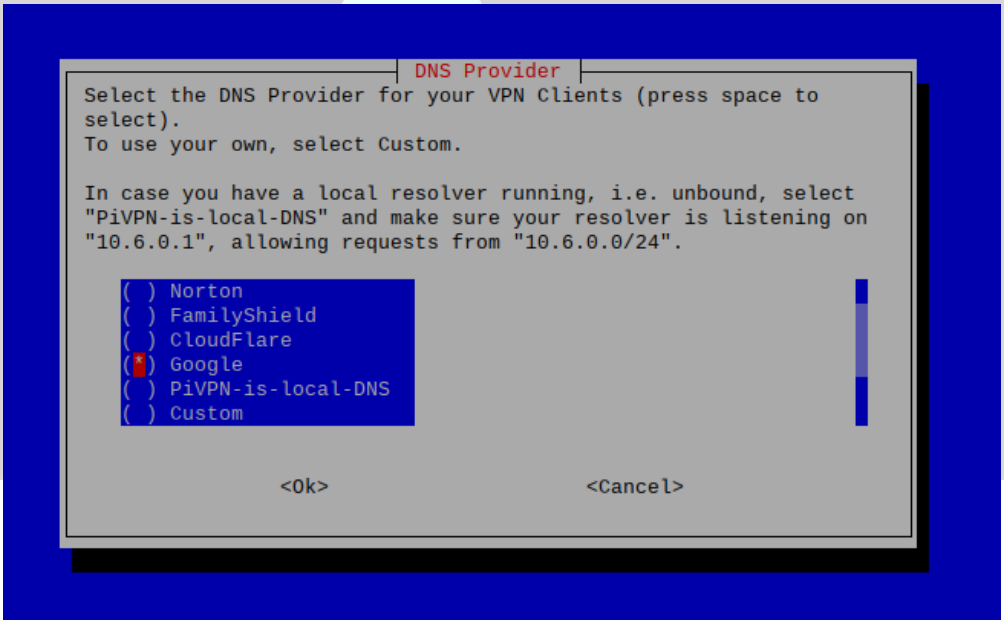


This phase was essential and required a port to be assigned, to host the WireGaurd server. By opening the port, it allows for Internet traffic incoming to the network, using the router. It was recommended to leave the port as default 51820.

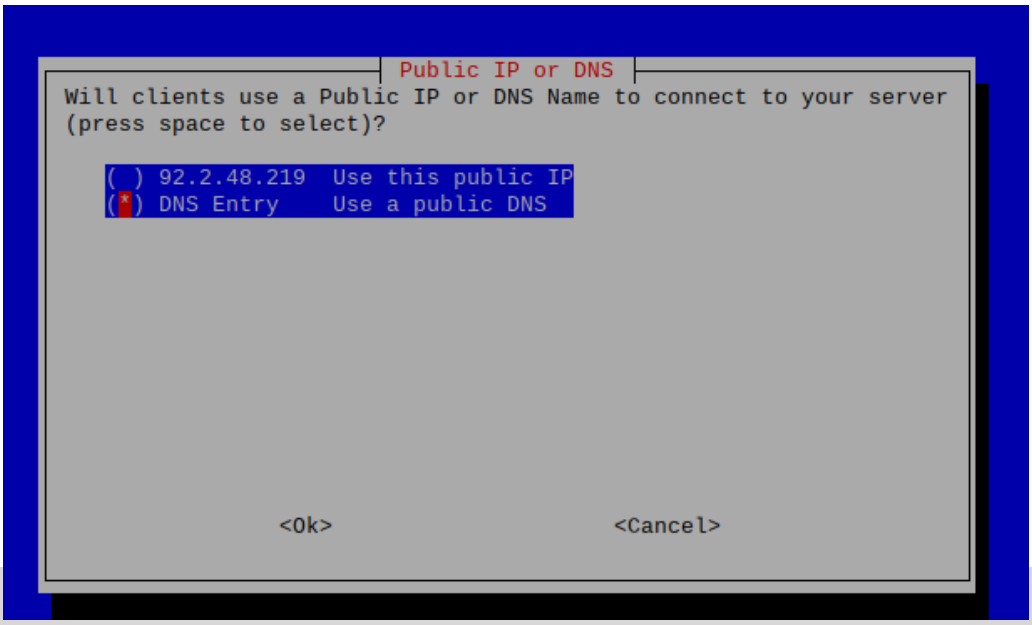


The image reveals choosing port 51820.

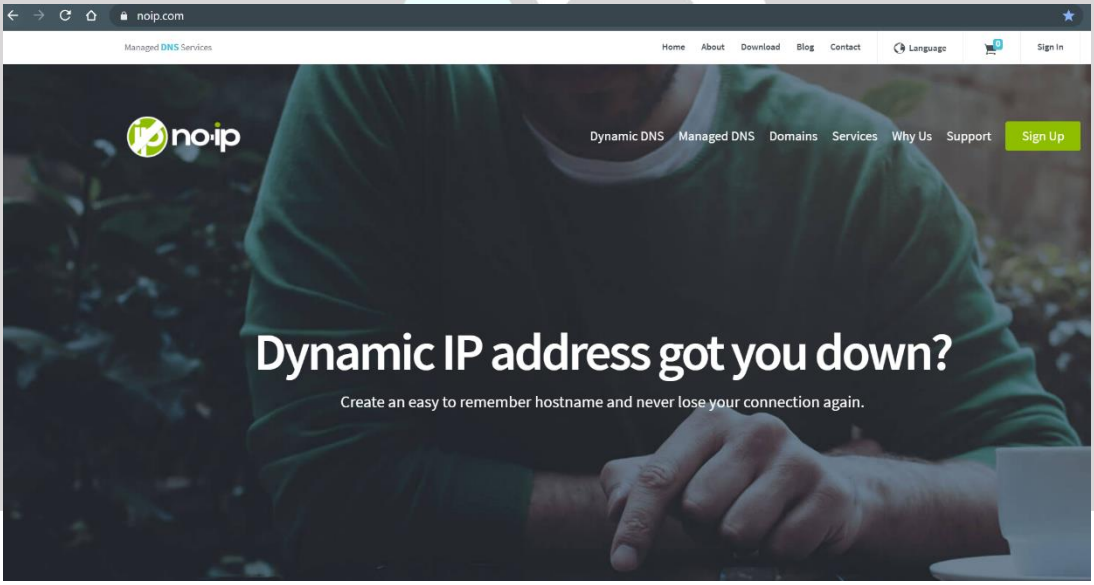
Next, the screen appeared as shown in the image below and required choosing a DNS service provider. In this instance, it was decided to choose Google on 8.8.8.8.



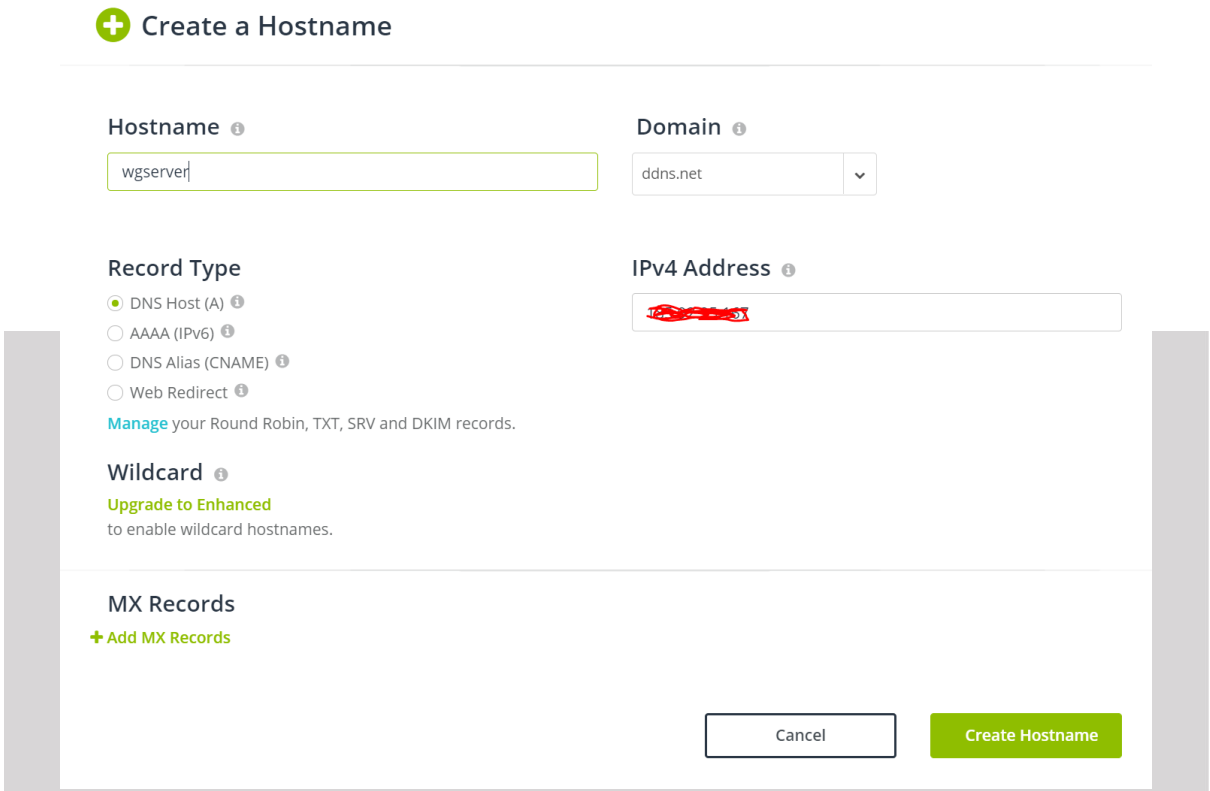
This phase required to select the Static IP address. Furthermore, the problem with using the public IP address, it switched each time a new connection was established to the network and therefore, would have affect the server.



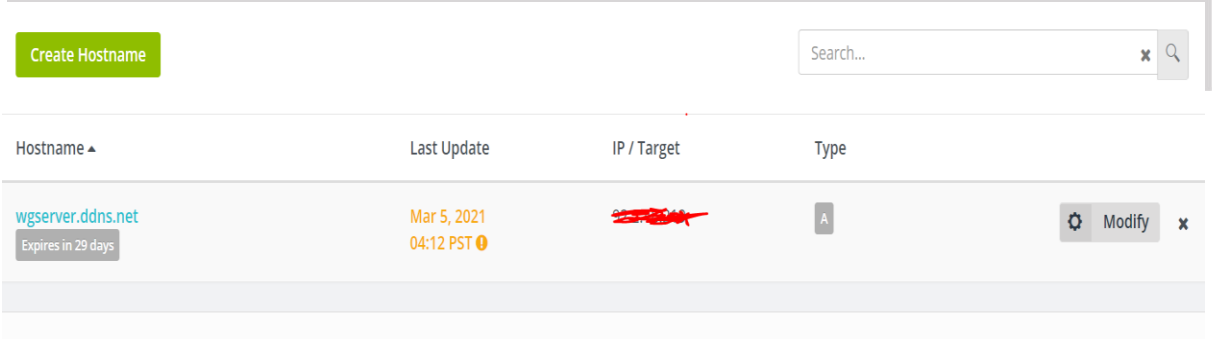
This phase required to access the following webpage, shown in the image below. After signing up, it would be required to create a new Hostname i.e. Static IP address.



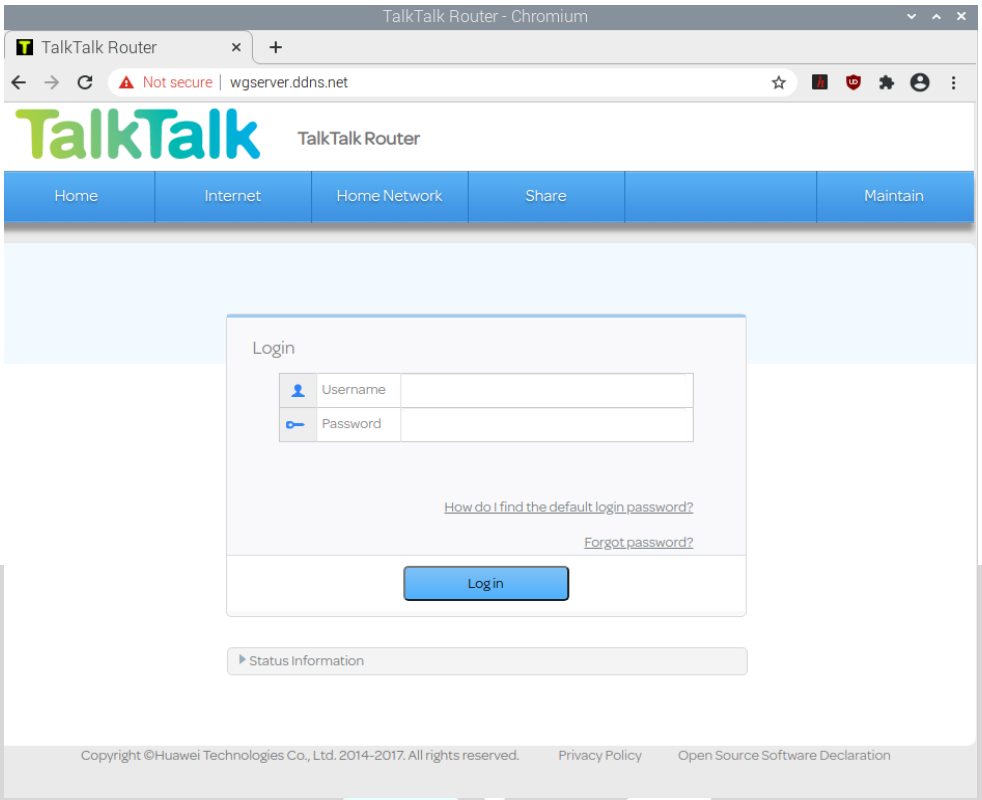
To create a new hostname, it was required to enter any name in the ‘Hostname’. Under Domain it was decided to keep it as default ‘ddns.net’. As you can see from the image below, it exposes the IPv4 Address, which is to identify the network interface on a device. After Creating the Hostname it were successfully created.



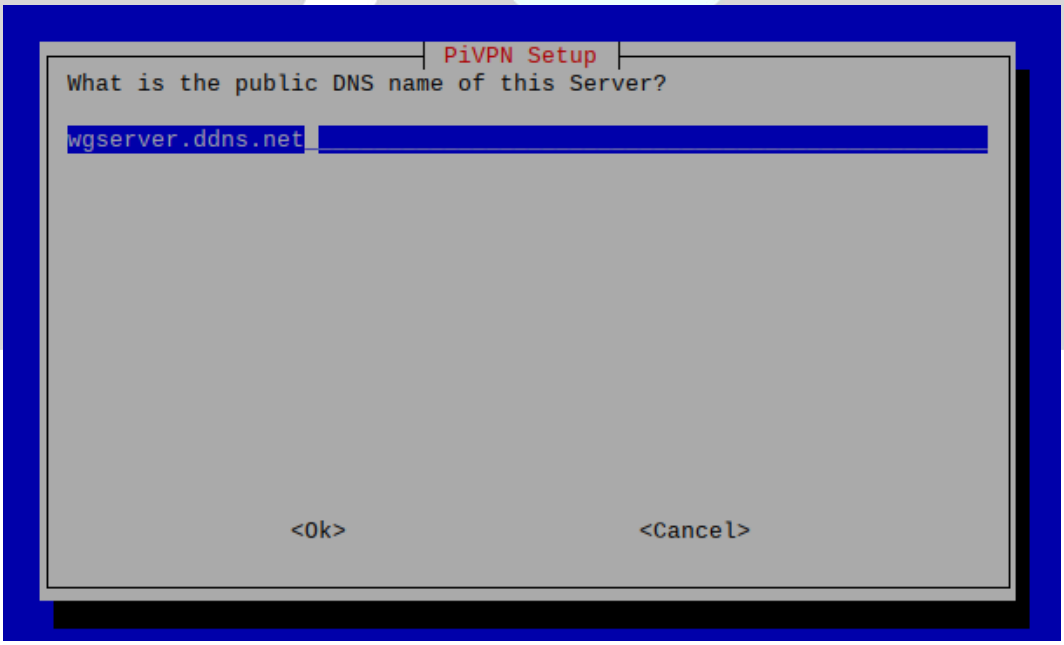
After creating the Hostname, it will be appeared as shown in the image below to indicate it was active. Furthermore, to test the Hostname was working, it had been decided to enter the Hostname in the URL address bar.



After entering ‘wgserver.ddns.net’ into the URL address bar, it exposed access to the default gateway on address 192.168.5.1 using the Hostname created earlier. This validated the Static IP address were set-up and working successfully. Next, it was required to go back into PiVPN GUI to proceed as follows.

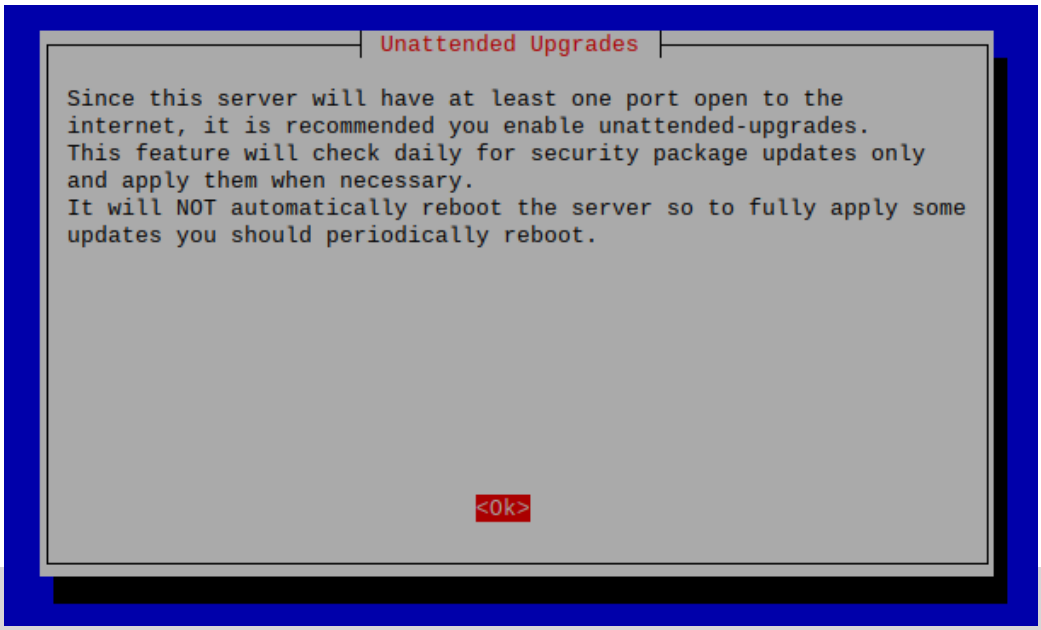


After setting up the Static IP address using NO-IP, it was required to enter the Hostname created previously, as shown in the images below.



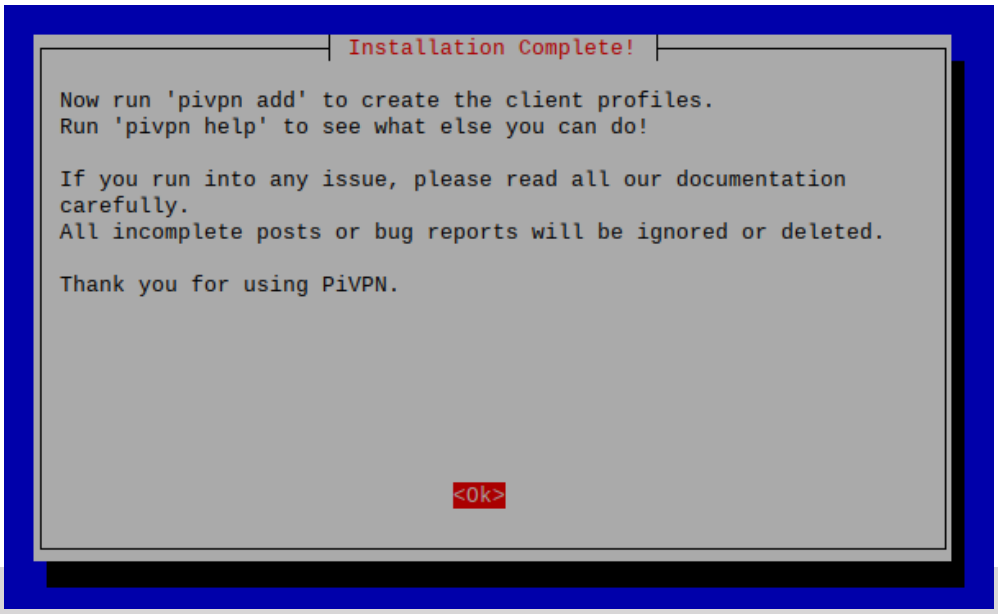
Next, PiVPN generated the server keys, Asymmetric Public and Private keys.

PiVPN recommended installing the latest security updates, as port 51820 is open and poses a potential threat, for incoming potential hackers.

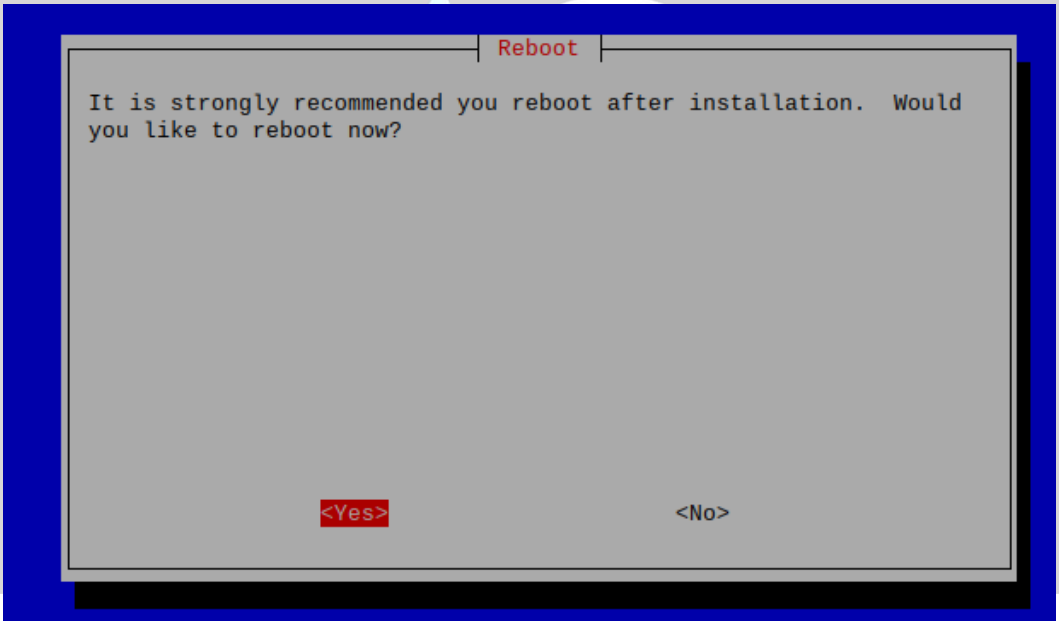


Installing the patches and proceeded as follows, as shown in the image below.

It was now required to reboot the Raspberry Pi 4 at this stage. After, will describe how to create client profiles and get connected to the server using a WireGaurd client from any machine.



The image reveals the installation had been completed.



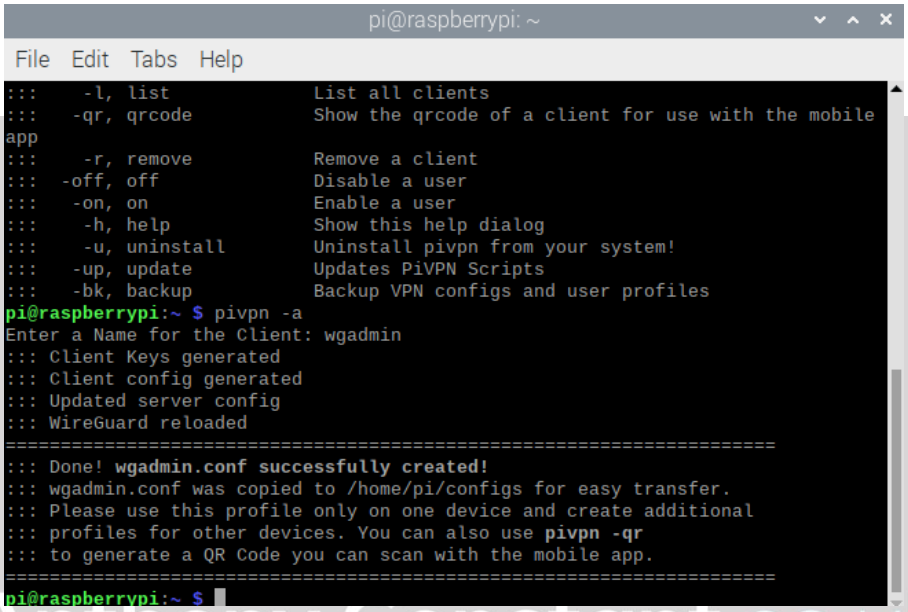
The image reveals rebooting the Raspberry Pi 4 to complete the configuration of the VPN server.

3.3.3 Setting up the user clients

Once successfully installing the WireGaurd server using PiVPN, it is essential to create the user clients to get connected. After the Raspberry Pi 4 was rebooted, it was required to open a terminal and type the following command to add a client:

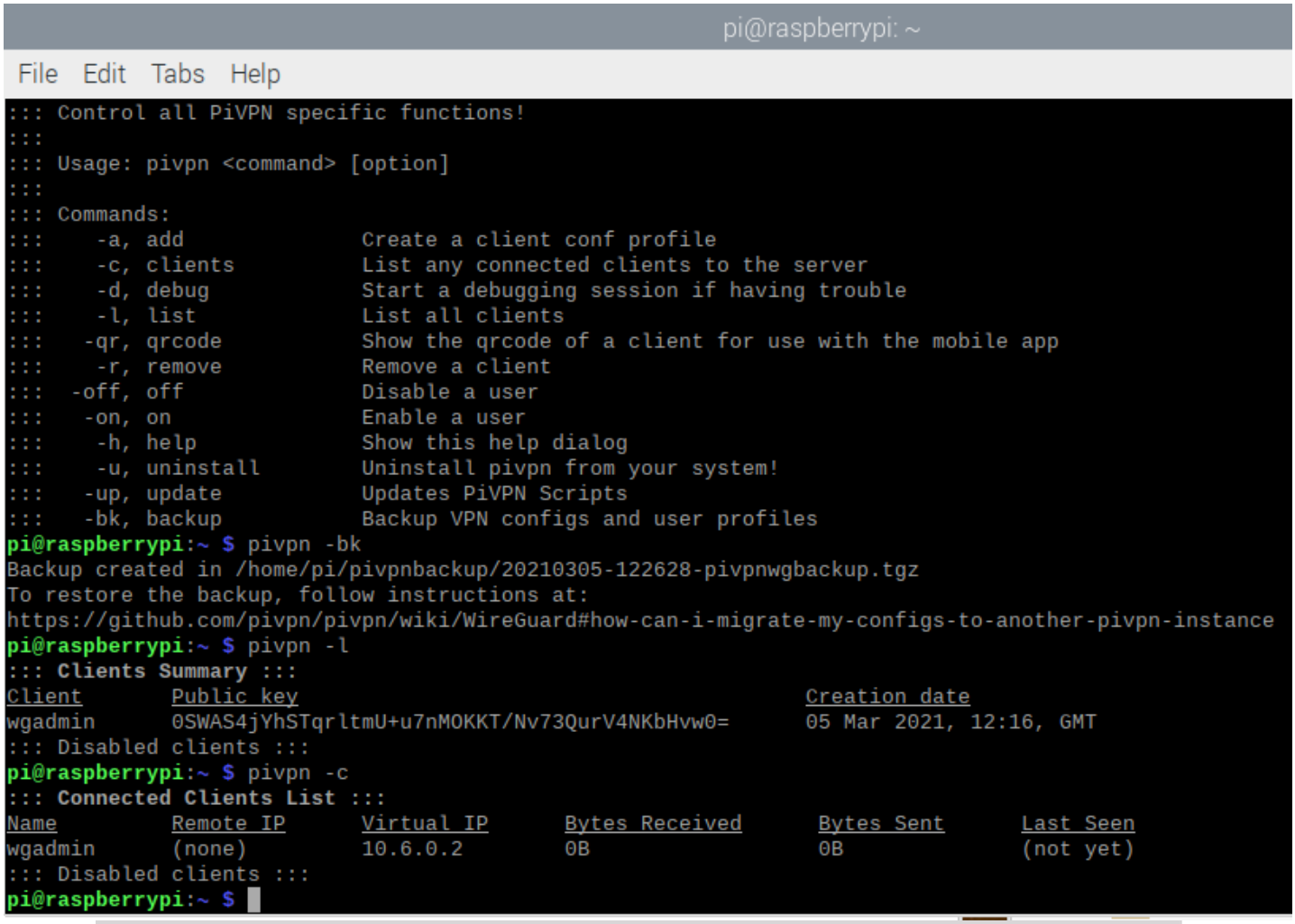
pivpn -a – adds a client

The terminal will request to enter a name for the client, in this instance it was called wgadmin. As a result, it produced the Client Keys and updated the WireGaurd server.



It was decided to run a command line as follows, to identify and validate the client had been added successfully.

- pivpn -l – lists all the clients
- pivpn -c – list the connected clients



The clients had been successfully added, as you can see from the image above. Furthermore, a configuration file was created and saved in location **/home/pi/configs**, which is used to import into the WireGaurd client on a machine, to connect to the server.

3.3.4 Testing: Connecting to the WireGaurd server

After the configuration file had been created, it was decided to transfer the config file onto a USB and plugged into a Windows machine. However, before importing the config file into the client, it was essential to investigate the config file itself using notepad, to gain a deeper understanding on the topic.

The configuration file is, as shown in the image above. From observing the [Interface], it is clear what had been exposed, from previous discussions within this project report.

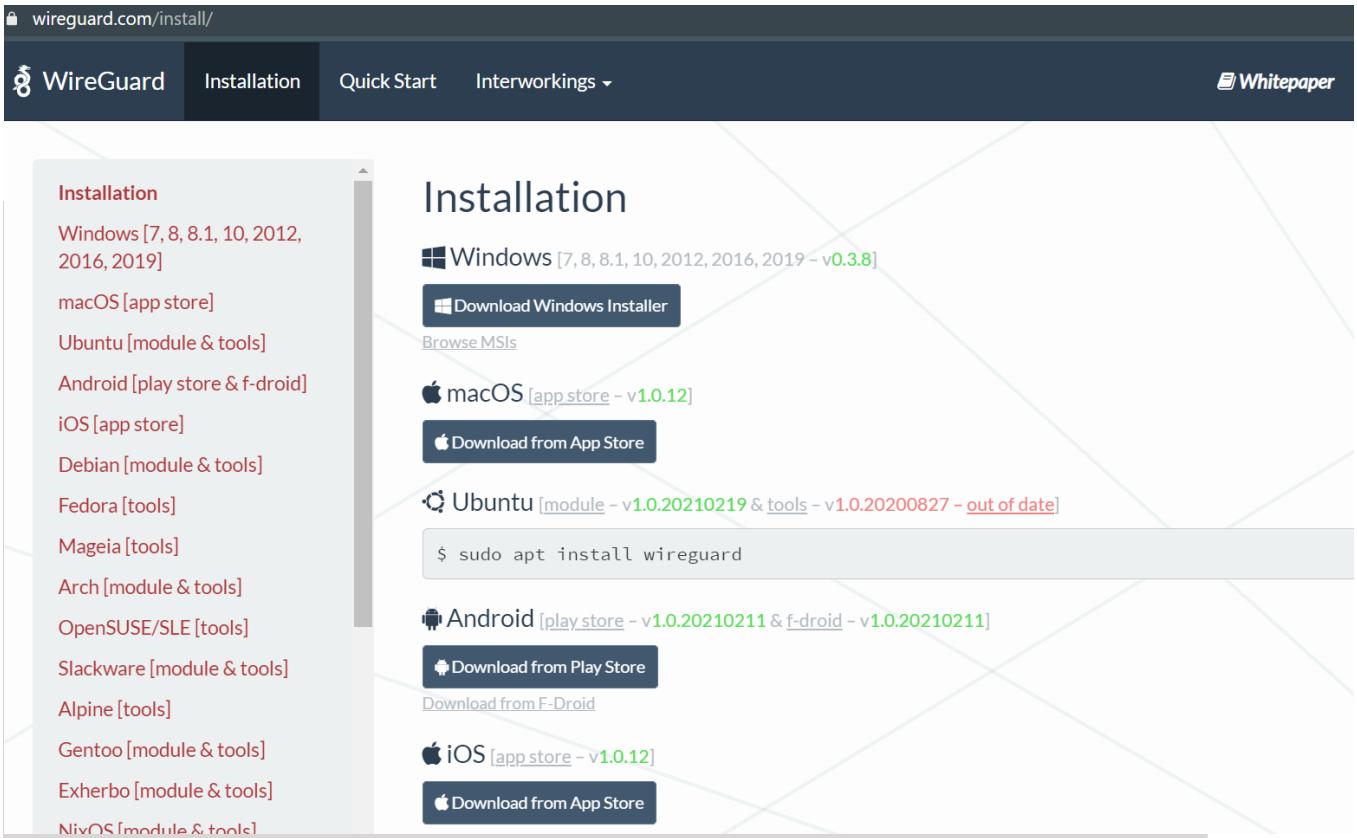
- The private key shown in the image above, was used to access the server from a client's point of view.
- The Address shown in the image, is the IP address provided to the client from the WireGaurd server.
- The DNS shows it's using Google's DNS 8.8.8.8.

From observing the [Peer], it can be described as follows:

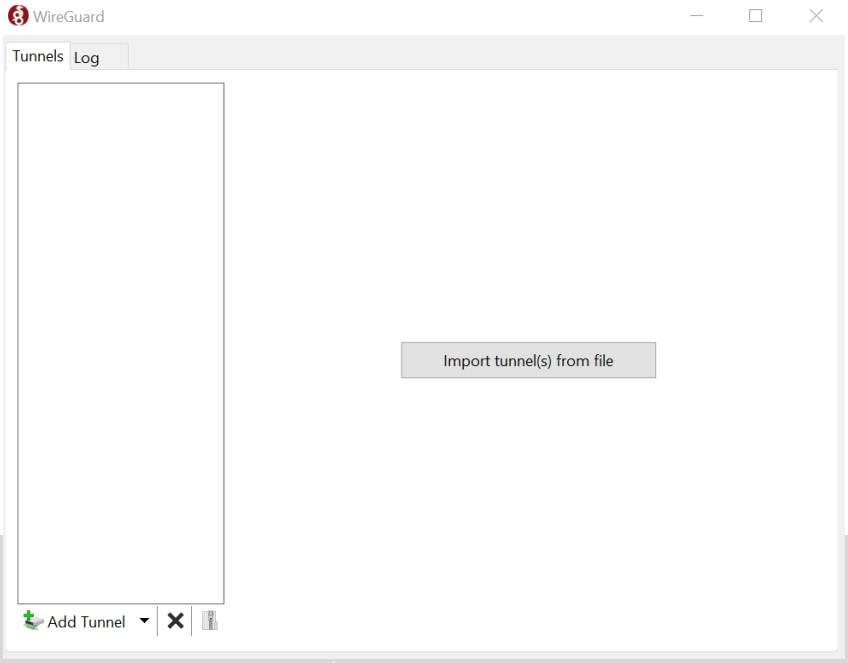
- The Public key is the server key generated by WireGaurd, which is publicly known to everyone. However, it requires the Private key to match the public key, to access the created WireGaurd server.
- A pre-shared key (PSK) is shown in the image above, which is expected to be acquired before receiving the private key for security.

- The Endpoint could be described as the Static IP address created earlier, using NO-IP. It also confirms its running on port 51820.

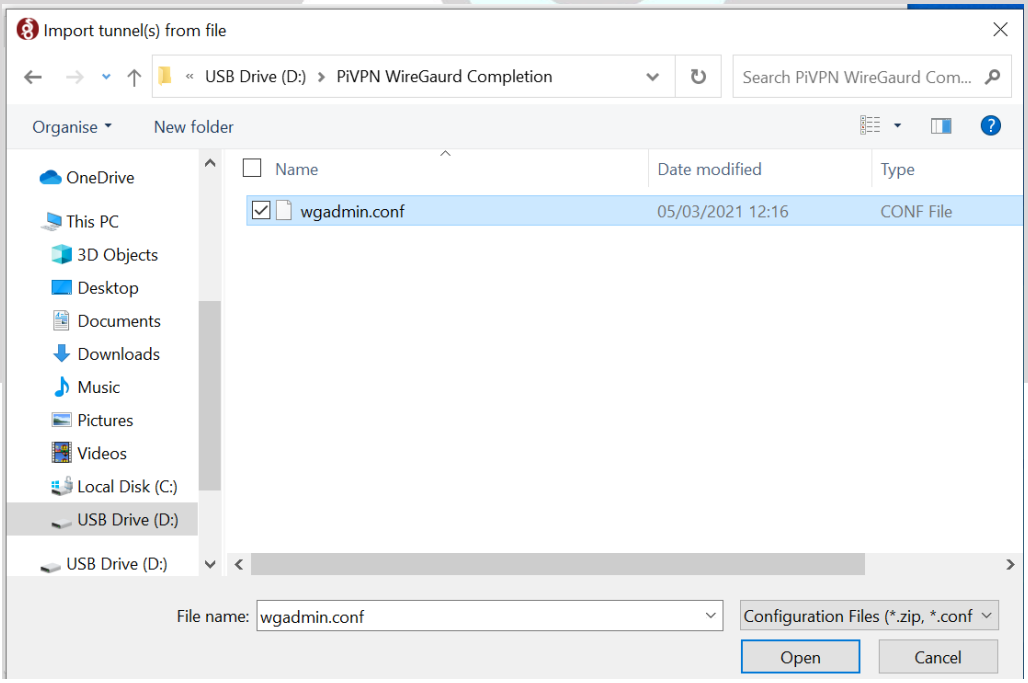
Once the configuration file was transferred onto the Windows machine, it was required to download and install the WireGaurd client using the Official website, which was provided within the references section.



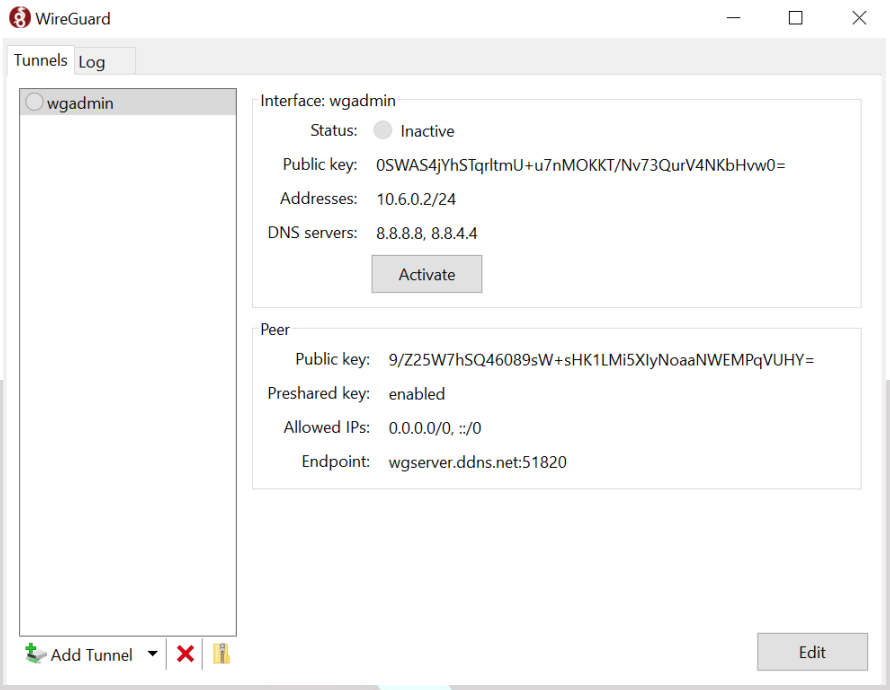
After installing the WireGaurd client which was a simple process using the .exe file provided, it was then required to select and import the config file into the client as shown in the images below.



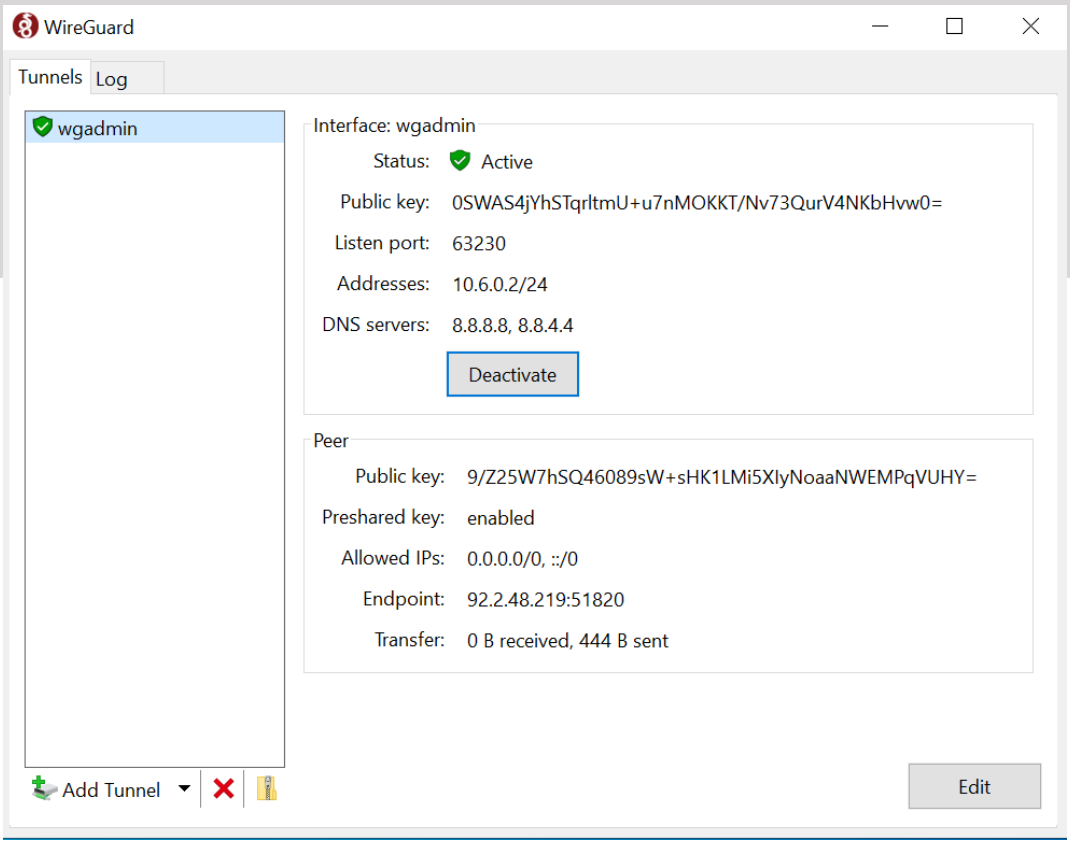
The image shows selecting the config file.



The image shows the config file had been imported.



The image shows the connection has been successfully established, after clicking 'Activate'.

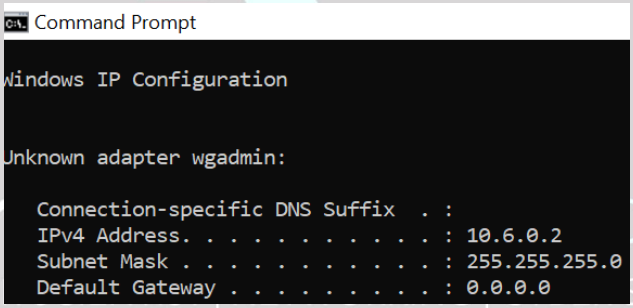


3.3.5 Conclusion

Overall, the tasks that were carried out within this report, has provided good insight, into the aspects of Networking and Cyber Security. Therefore, the brief has been met, in terms of gaining a deeper understanding into these topics.

Eventually, after deploying the WireGaurd VPN server, it could be said that it has successfully been used to overcome some Internet privacy and security issues.

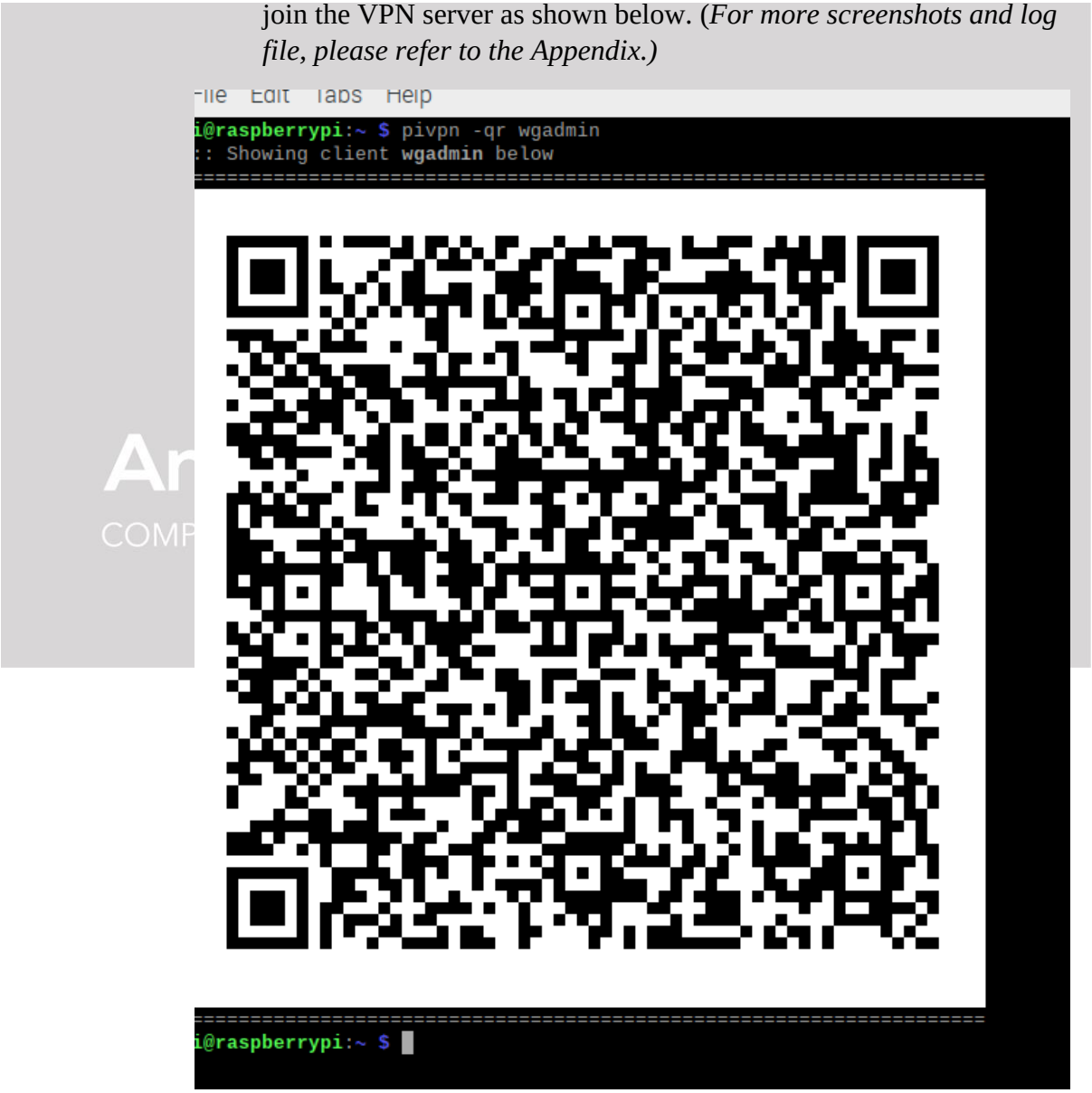
By Implementing a WireGaurd server, it could be said that as a result, have overcome Internet security issues entirely. Once connected to the WireGaurd server, a virtual tunnel was created such that, it provided the user client with a new virtual IP address given by the server, which is highly secure and impossible to intercept the connection. i.e. man-in-the-middle attack(MitM) attempts. A virtual IP address had been assigned, once importing the config file into the client. This was performed on a windows machine within command prompt, by running the following command 'ipconfig' as shown in the image below.



However, as mentioned previously, when WireGaurd assigns the IP address to a client, it stores the IP address on the server for long periods of time or until rebooted. This makes WireGaurd run much faster than any other VPN however, it poses a potential privacy threat, which leads into the next part of the discussion, has setting up a WireGaurd server overcome privacy?

As mentioned, WireGaurd poses a potential privacy threat and was not designed for privacy but is highly secure. Whilst WireGaurd by default stores the clients IP addresses and connection times on the server, it is possible to implement a solution to this privacy issue. For example, NordVPN offer their own interpreted versions of WireGaurd protocol that work around the IP address problem and as a result, no logs are stored. In the future it is intended to implement a similar solution, to ensure privacy. However, it is extremely unlikely for someone to hack into the network and obtain the IP address within a household server and therefore privacy is kept within its limitations.

What I have learned from this whole experience is how to set-up a WireGaurd server using PiVPN. I have also learned how to develop a WireGaurd server using the correct networking configurations and explain how WireGaurd works in extreme detail. In this process, I have learned the stages someone must go through to set-up and maintain your their WireGaurd VPN server, what each stage involves and what tools are used. For example, the prerequisites phase was a good insight in finding out the different tools which could be potentially used. The task consisted of gathering resources using the Internet, with different keyword searches. Additionally, maintaining the VPN server was insightful to looking for further solutions, such as hosting which revealed more detail into the log file and how it worked. Furthermore, a QR code had been revealed with the intension for users to scan it and join the VPN server as shown below. (*For more screenshots and log file, please refer to the Appendix.)*



4.0 Chapter 2: How to build a PIR motion detection sensor

In this part of the report it will be discussed the second phase of building a PIR motion detection sensor, to detect potential real-world threats.

4.1 Getting started how to build a PIR motion detection sensor

As part of building the PIR motion detection sensor, it consisted of gathering the following resources discussed within the literature review section. However, in the next part, exposes the list of prerequisites to build it.

4.2 Prerequisites (Complete working artefact refer to Final Attempt.)

Name	Component	Description	Comments
Hardware	HC-SR501 Passive Infrared Sensor(PIR) Motion Detection Sensor Module	A device in which can detect change in the infrared light. Uses PIR to detect activity within its range of view.	the link to the module is provided here *(Amazon, 2021)*
	ELEGOO UNO R3 Development Board	To be used as a micro-controller which connects to any machine through USB and the PIR motion sensor	Uses ATmega328 like the official Arduino UNO R3 board. The link to the module can be found here *(Amazon, 2021)*
	Jumper Cables (Male to Female M-F) X3	To connect the PIR motion sensor to the micro-controller board	The link to the module is provided here *(eBay, 2021)*
	Windows 10 Machine	To be used to connect the micro-controller and install software/run code.	It is recommended to use Windows 10 to prevent any combability issues.
	HDMI monitor, Keyboard, Mouse	To be the Windows 10 machine peripherals	
	USB Cable	To connects the micro-controller to the Windows 10 machine	
Software	Arduino IDE	An IDE in which code can be developed and uploaded to modules such that, it controls the behaviour of the module using the code	The official website for Arduino IDE can be found here *(Arduino, 2021)*

4.3 First failed attempt: How to build a PIR motion detection sensor

The first attempt of building the PIR motion detection sensor were unsuccessful however, it is almost inevitable whilst working with hardware that problems will arise. It is essential to learn from these failures and develop a new solution to tackle the problem, which ultimately led to further research and acquiring additional hardware.

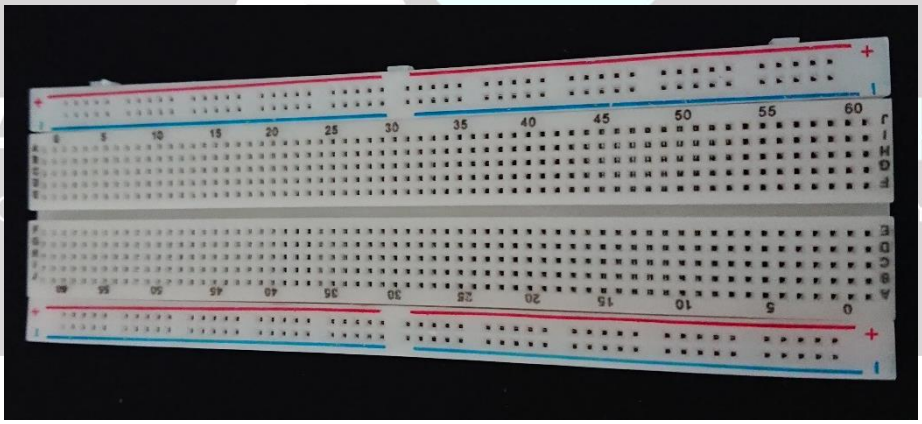
4.3.1 Acquiring and connecting the hardware components

Within this attempt, the following hardware were required:

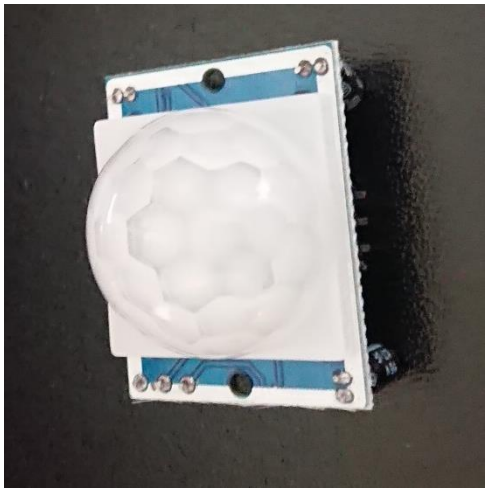
- Breadboard
- Jumper cables
- Resistors
- LED Bulbs
- Raspberry Pi 4
- PIR motion detection sensor

After studying the different hardware components and how they worked together, it was then required to go ahead and acquire the different components, as shown in the images below.

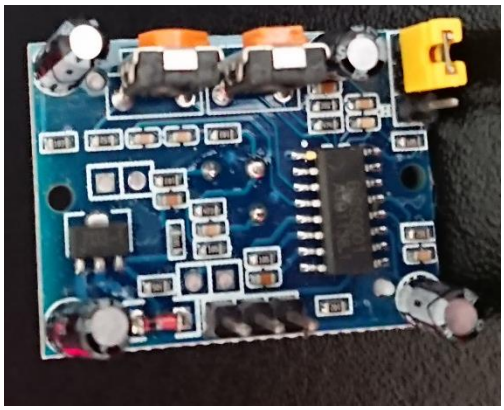
Solderless Prototype Breadboard



Front - PIR Motion Detection Sensor



Back – PIR Motion Detection Sensor



Resistors to prevent short-circuit

LED bulbs to indicate motion detected

Jumper Cables to connect everything

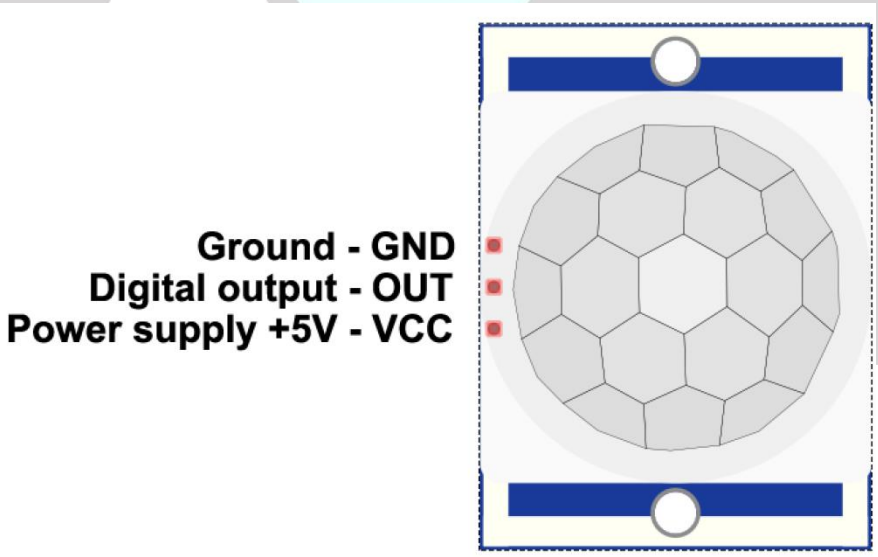
After acquiring the components, it was then required to connect the different components together. It was first essential to connect the PIR motion sensor to the breadboard, using male to female(M-F), following another set of MF cable going into the Raspberry Pi 4.

Note: a female end went into GPIO 1, then connected another female end into GPIO 3 then, connected another female end into GPIO 4 and lastly, connected female GPIO pin 6. (Raspberry Pi 4)

After connecting the jumper cables into the Raspberry Pi 4, next was required to connect the male pins into the breadboard as well as the PIR motion detection sensor.

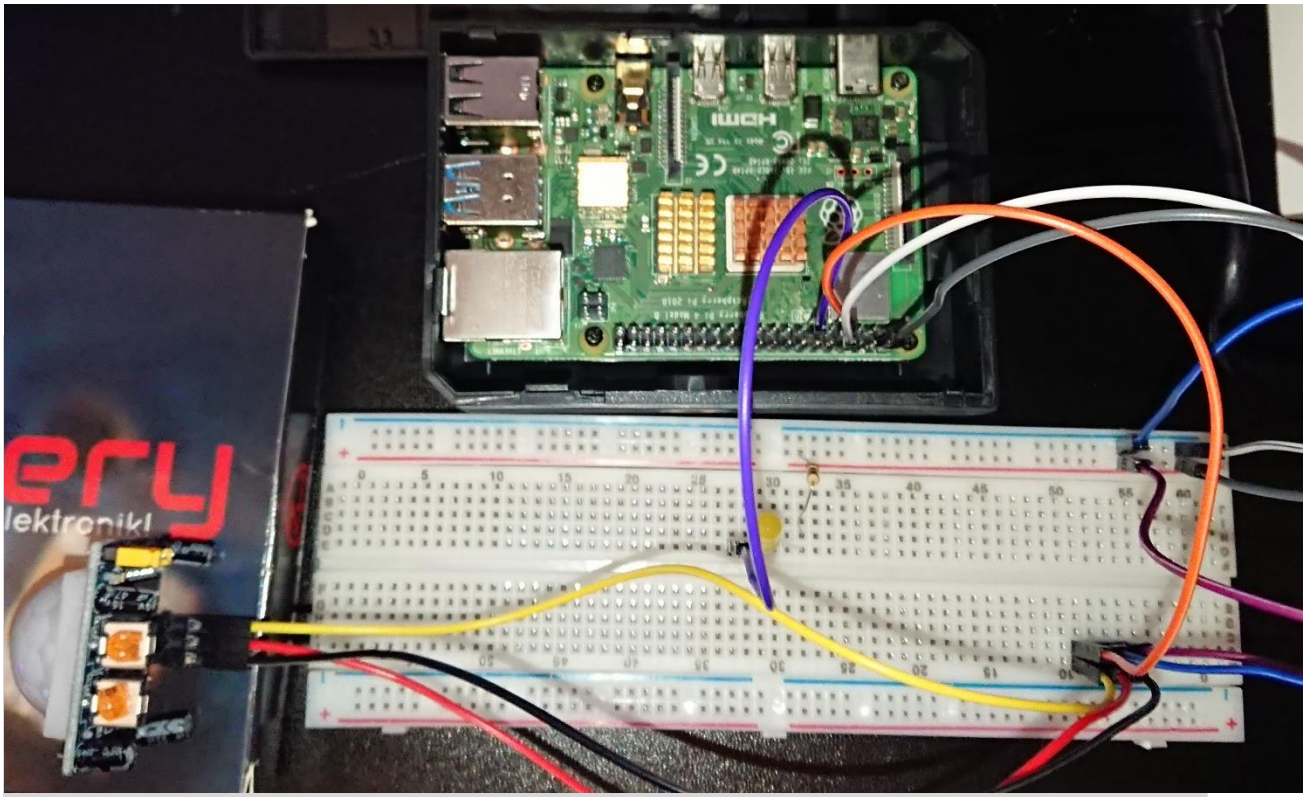
Note: GPIO pin 4 went into the digital output, in line with the PIR sensor on the breadboard. Then, connected GPIO pin 1(5v) into the positive terminal on the breadboard followed by, GPIO pin 3(ground) into the negative terminal.

The image reveals which each pin indicates on the PIR motion detection sensor. The Digital output is what sends the signal output to the Raspberry Pi 4 when it has detected motion, as well as getting power from the Raspberry Pi 4 using 5V and grounded using negative.



The diagram was obtained from: A-Z Delivery Whitepaper

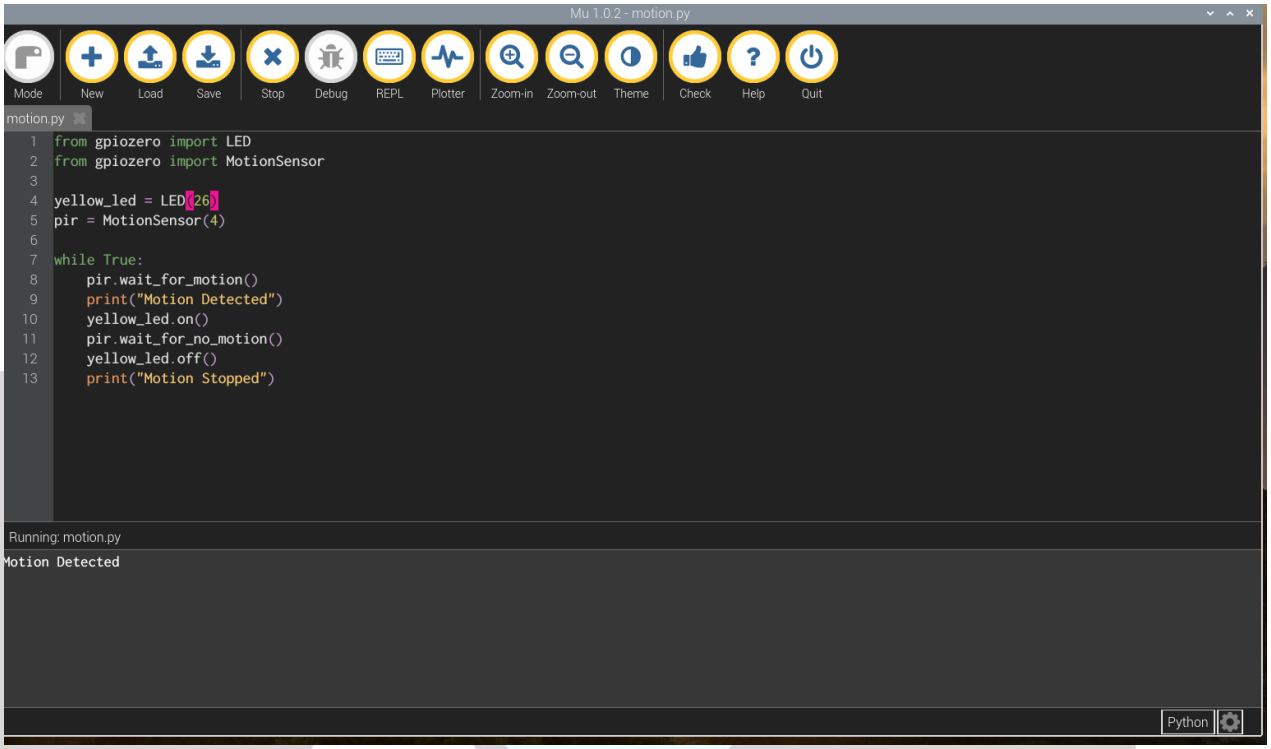
After connecting the jumper cables from the Raspberry Pi 4 into the breadboard, it was then necessary to connect the resistor into the negative rail and the other end into the breadboard(26). Then, connected the LED bulbs short end in parallel with the resistor and long end parallel to GPIO pin 6 from the Raspberry Pi 4.



After connecting everything it was then required to boot up the Raspberry Pi 4 and create some code, to control the behaviour of the PIR motion detection sensor.

4.3.2 Testing: Using MU IDE to test the PIR motion sensor

After booting up the Raspberry Pi 4 and everything were connected from the breadboard, next was to open ‘MU IDE’ to create the code, as shown in the image below.



It was essential to import the ‘LED’ and ‘MotionSensor’ modules from the library. Next, was to define the GPIO pins its connected to and specify which module. Then, a basic while loop to print “motion detected” and turn LED on such that, it should be turned off and await motion when no motion has been detected, as you can see from the image above.

After running the script, it had printed out “Motion Detected” once and nothing else happened after that. Therefore, tried different solutions and attempted switching the cables, in the instance they could have been inverted. However, after switching the cables it was still unsuccessful and Ultimately, led to conducting further research on how to solve the problem. It could be said that after conducting further research, the micro-controller was discovered and eventually acquired, as the next attempt reveals. (*For more screenshots, please refer to the Appendix.)*

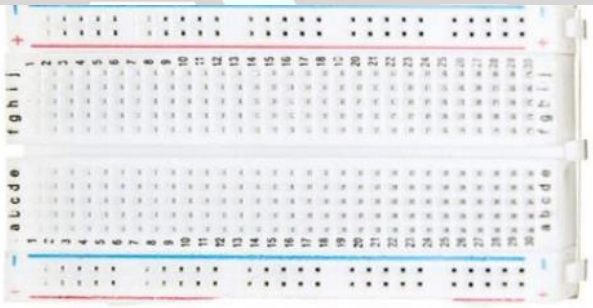
4.4 Second failed attempt: How to build a PIR Motion detection sensor

The second attempt of building the PIR motion detection sensor were still unsuccessful however, after the first failed attempt, it could be said that the microcontroller was discovered and had progressed further into building a system to detect potential real-world threats.

4.4.1 Acquiring the hardware components and installation/configuring Arduino IDE 1.8.13

After conducting further research, some new evidence *(Binary Updates, 2018)* was found and as a result, acquired the following additional hardware:

- Jumper Cables Male to Female(MF) x3
- PIR Motion sensor
- Breadboard
- ESP32-WROOM-32D Microcontroller Development Board



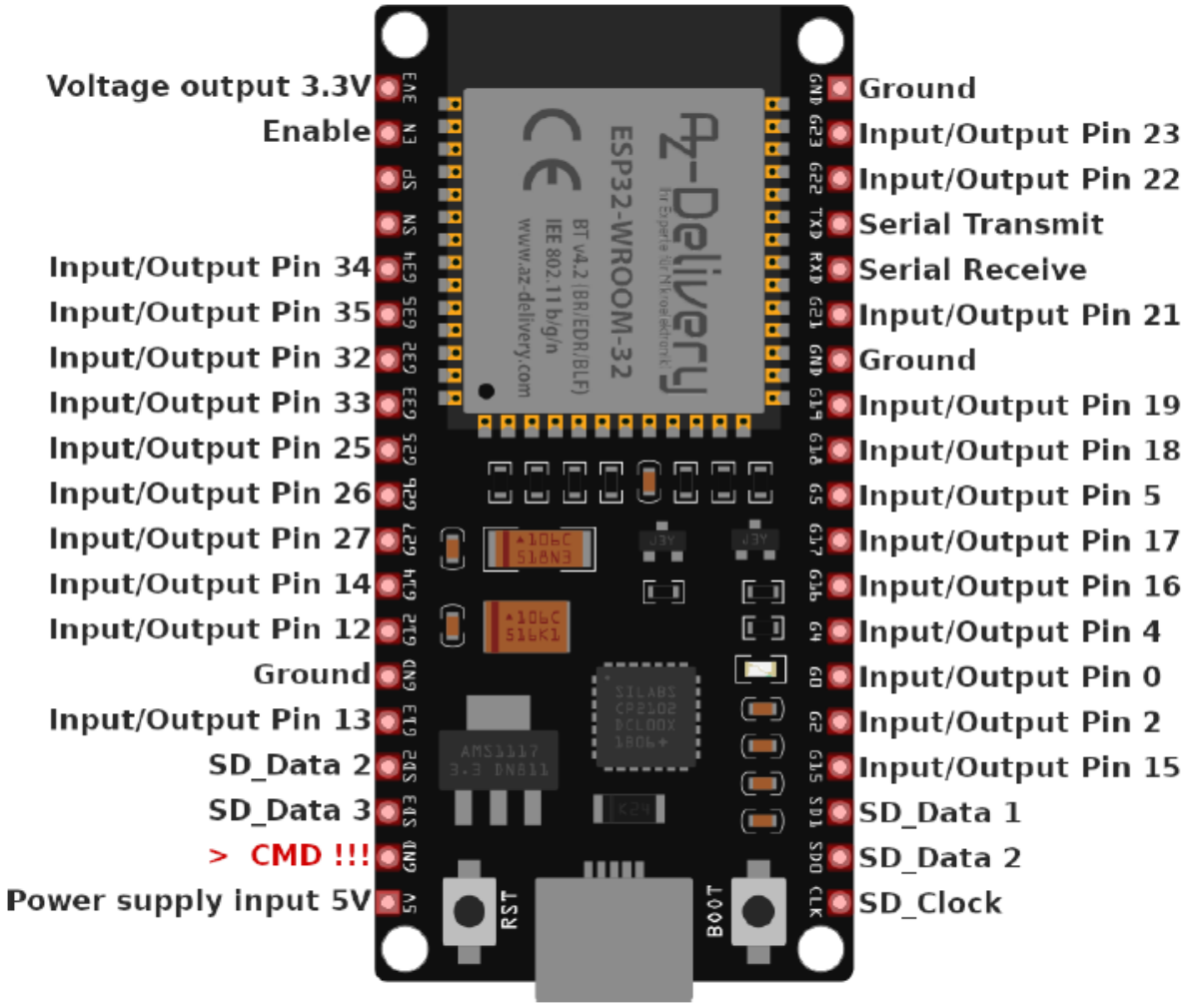
The ESP32 Microcontroller is a development board created around the ESP32 chip itself, and contains a 3.3V regulator, a USB programmer circuit for the ESP32 chip and more. It also comes with pre-installed firmware which will enable the programming of the board using programming language, sending the commands via serial port and can

be used with various IDE's however, for the purpose of this project report, will be using Arduino IDE.

It operates at 3.3V and the PIR Motion sensor operating at 5V and therefore, will need to be modified. Also, the ESP32 will be used as the main microcontroller, the PIR Sensor as the motion detection sensor and breadboard to connect them both.

It is specifically designed to only work on a solderless breadboard, as shown in the image above. After learning about the ESP32 Microcontroller and how it connected to the PIR motion sensor, it was essential to go ahead and begin connecting the components together. (*For full specifications, please refer to the Appendix)*

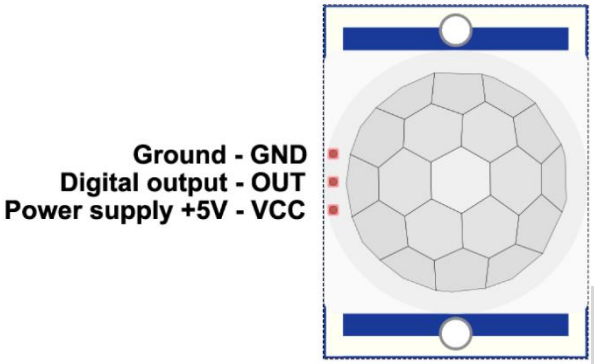
The ESP32 has a total of 38 pins and each pinout serves for a different purpose, as shown in the image below.



The diagram was obtained from: A-Z Delivery Whitepaper

Before connecting the PIR motion sensor to the microcontroller, it is essential to understand the PIR motion sensor further, than the

literature review, to ensure everything has been covered within the topic. When the crystalline within the sensor is exposed to some infrared light, the voltage output is converted along with the infrared light intensity such that, the voltage output from the sensor gets amplified and sent to the microcontroller to ultimately detect motion. The pinout is shown in the image below:

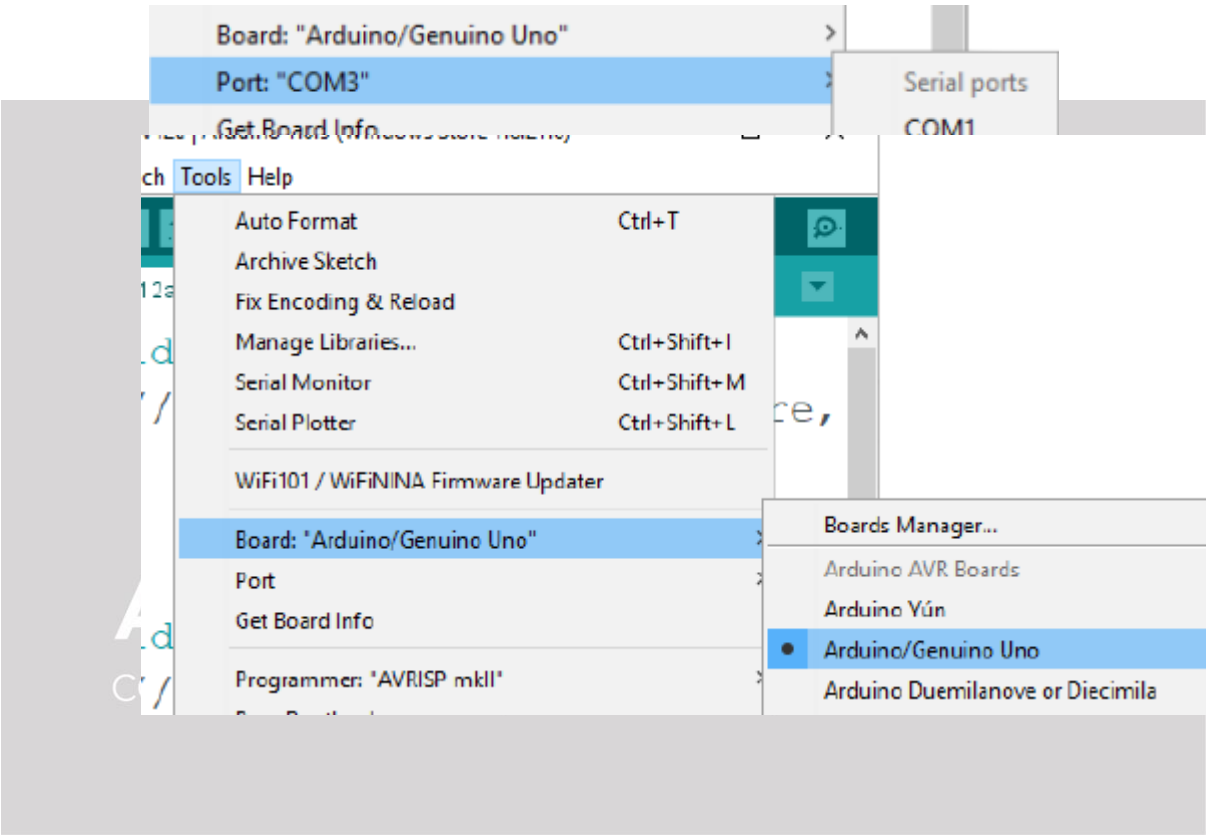


The diagram was obtained from: A-Z Delivery Whitepaper

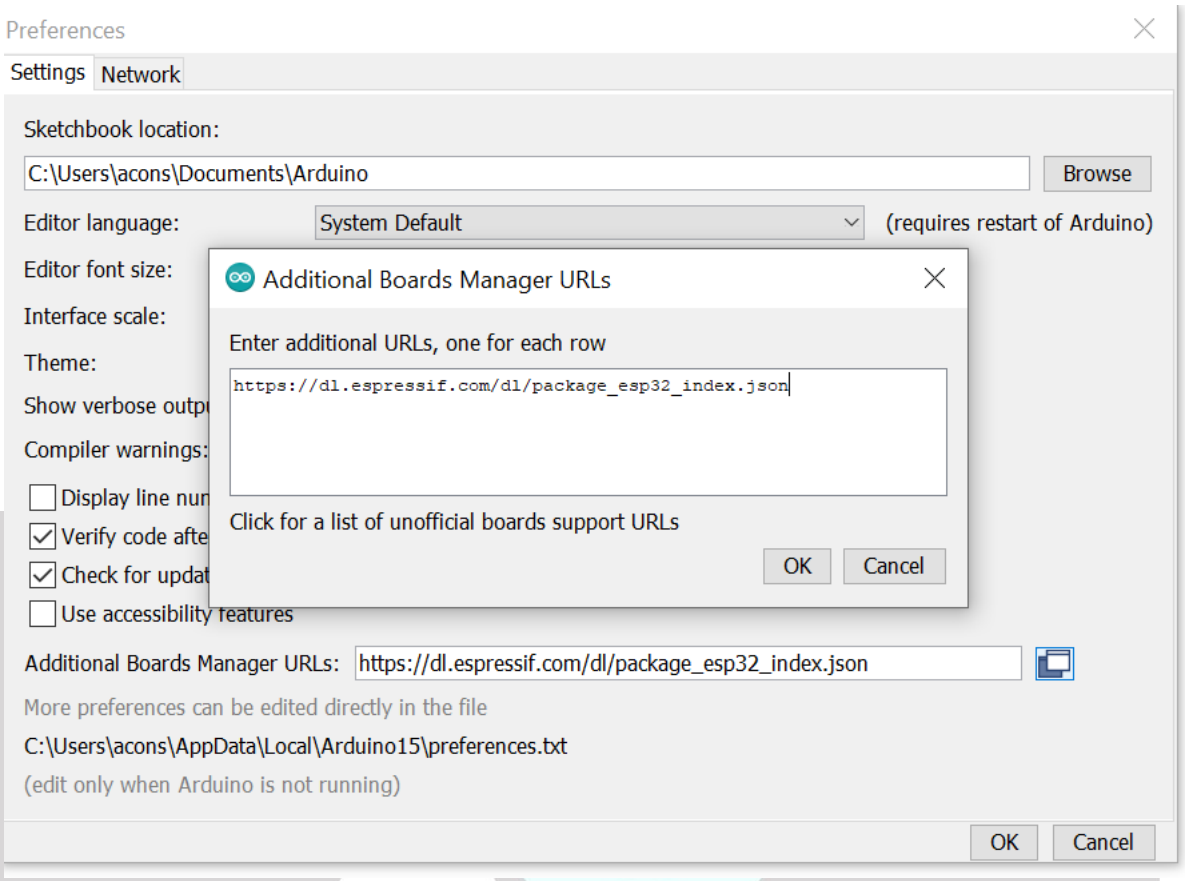
The next step was to install Arduino IDE by going to the official Arduino website using the following link https://www.arduino.cc/ *(Arduino, 2021)* using a Windows 10 machine, as shown in the image below.

Upon installing Arduino IDE using the .exe file next, was to run the application from desktop and edit the settings accordingly.

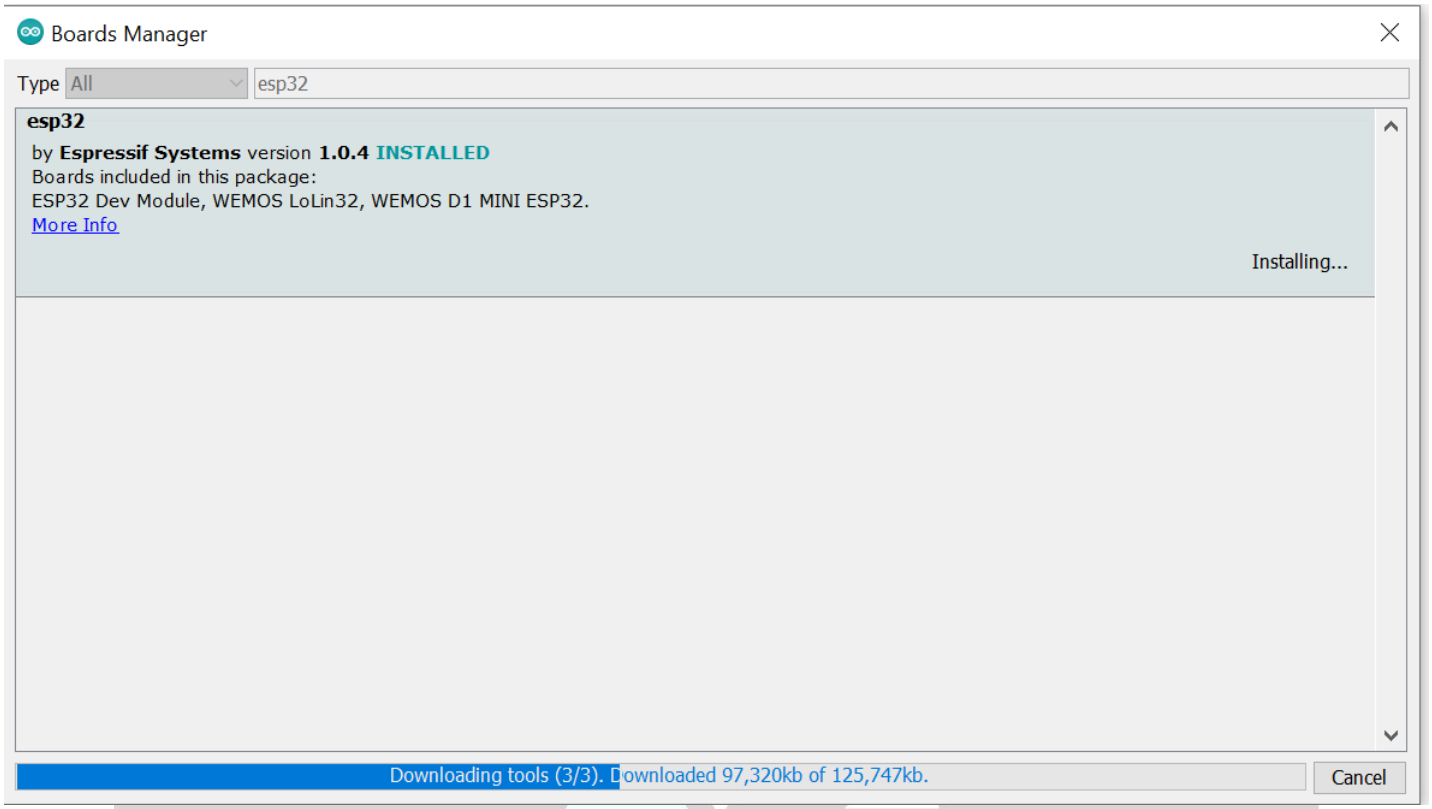
After specifying the board type ‘Arduino/Genuino Uno’ in Tools>Board, it could be said that the port was required to be specified, as shown in the image below.



After specifying the port (COM3) the microcontroller was connected to the Windows machine, it was then required to download and install the ESP32.json package using the following URL: https://dl.espressif.com/dl/package_esp32_index.json *(Espressif Systems, 2021)*



After pasting the URL into File>Preferences>Settings and clicking the small blue/white icon in the image above, then clicked ‘OK’. Next, was to install the board from ‘Boards Manager’.

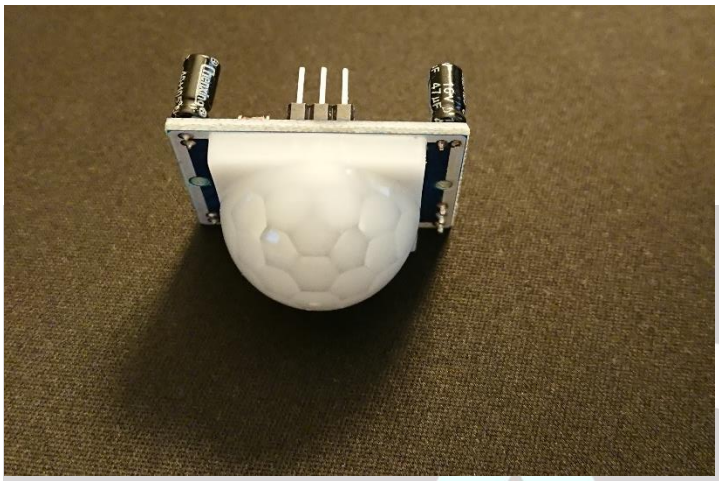


After proceeding to Boards Manager within Arduino IDE, it was then essential to type the name of the microcontroller chip ‘ESP32’ to download and install the package provided by Espressif Systems version 1.0.4.

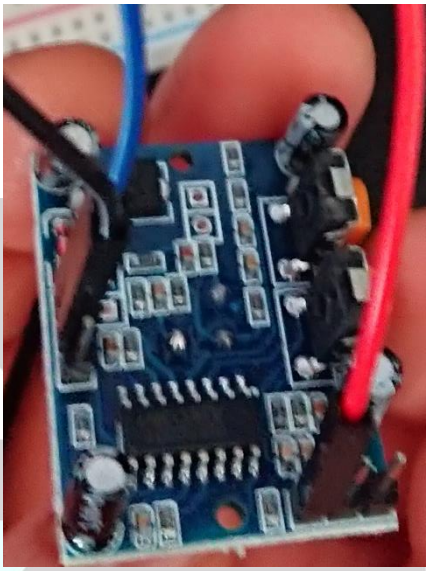
4.4.2 Connecting the hardware components

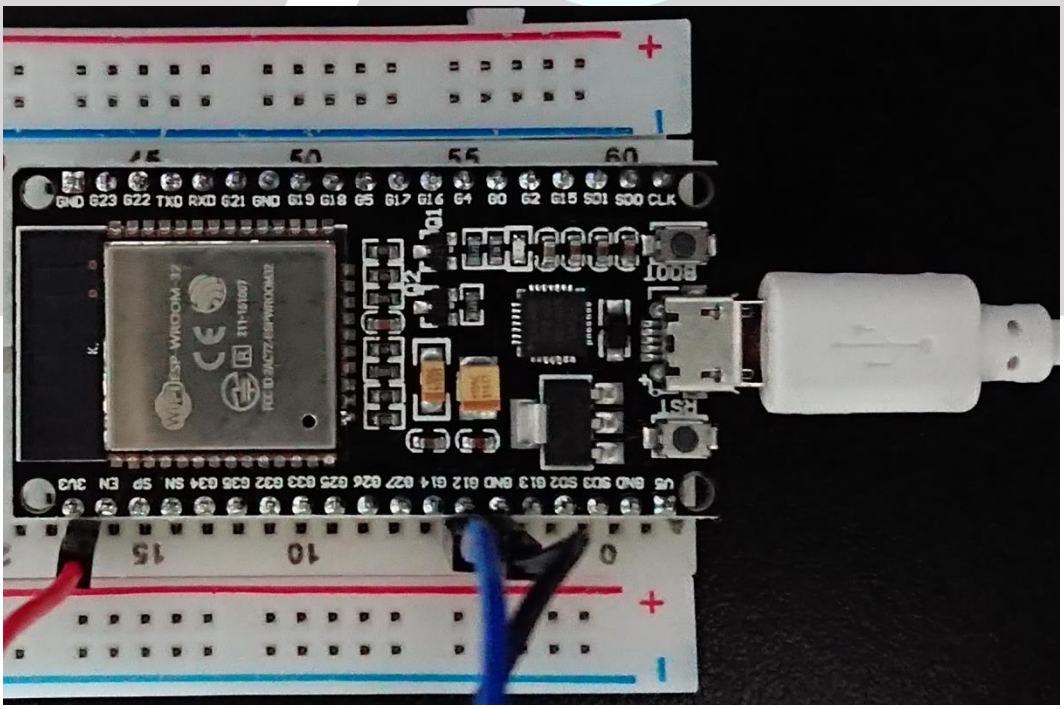
After several minutes it had finish installing and navigated to Tools>Board then specified the new installed board name ESP32 Dev Module. The settings had been correctly set-up regarding Arduino IDE, next was to connect the PIR sensor with ESP32 microcontroller using the breadboard, as shown in the images below.

PIR Sensor



PIR sensor with jumper cables



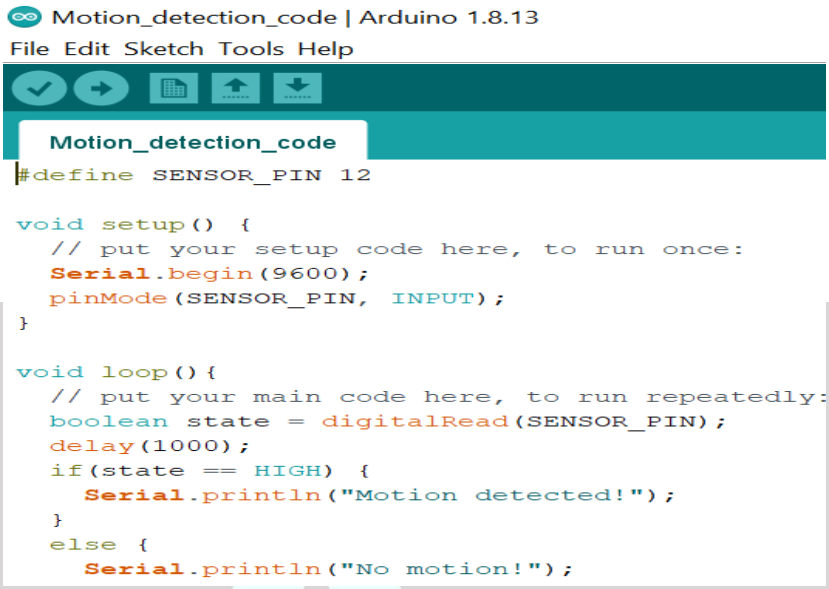


The red wire was connected to the 3V3 then, the black wire was connected to Ground and the blue wire connected to G12 on the ESP32 microcontroller.

PIR Motion Sensor Pin	ESP32 microcontroller board	Wire
OUT	G12	BLUE
GND	GND	BLACK
H PIN	3V3	RED

4.4.3 Testing: Developing the code in Arduino IDE to test the PIR motion sensor

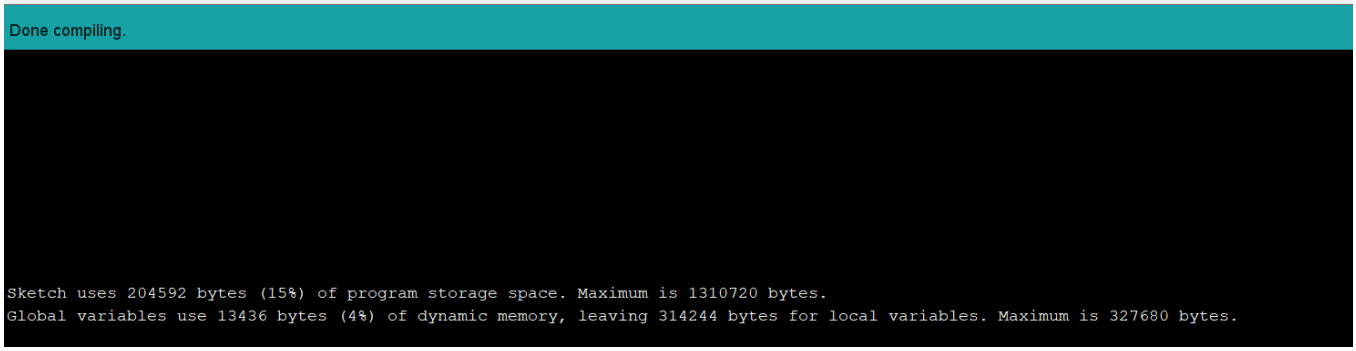
Upon connecting the PIR sensor and ESP32 microcontroller to the breadboard, next was to develop some code within Arduino IDE, to control the behaviour of the PIR sensor when it detected motion.



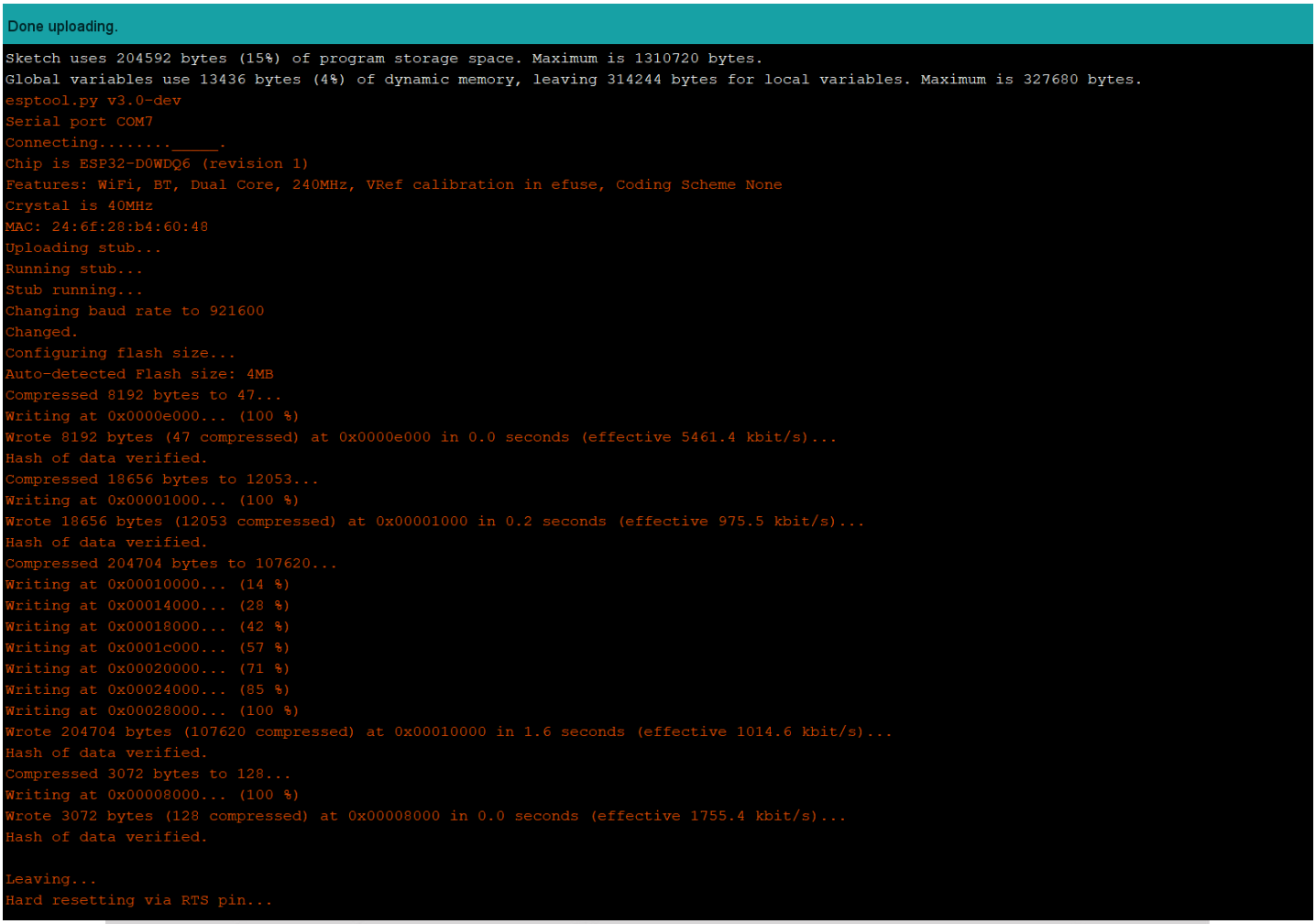
The code created above was simple enough, to test if the module was going to be working successfully or not, before developing the full code for PIR motion sensor to detect potential real-world threats.

It was essential to define the PIR sensor pin(OUT) from G12 on the microcontroller using the breadboard. Then a void function was setup and used to run the PIR sensor. Next, a loop was created and if the PIR sensors state was set to HIGH, which meant motion was detected, then print a message saying “Motion Detected” otherwise, the state is LOW and print out “No motion”.

After developing the test code, it was required to compile and upload the code to the microcontroller connected via USB. To upload the code, it was essential to click the upload icon in Arduino IDE, whilst holding down the 'BOOT' button on the ESP32 microcontroller, as shown in the images below.



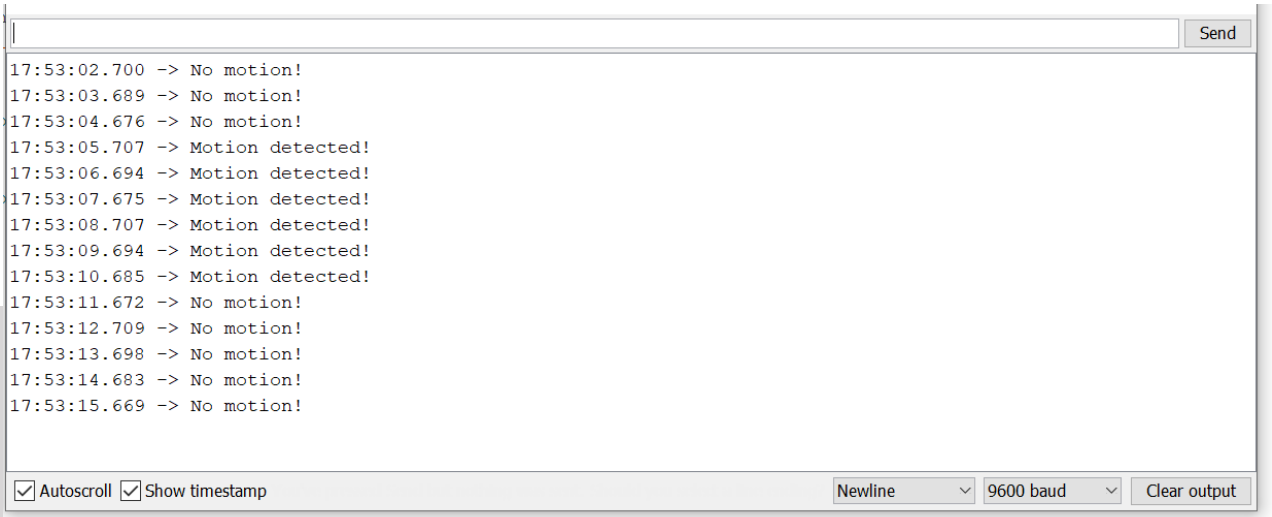
The image reveals the code had been compiled successfully.



The image reveals the code had been successfully uploaded to the ESP32 microcontroller.

4.4.4 Conclusion

After uploading the code to ESP32 microcontroller it was then required to run the Serial Monitor by going into (Tools>Serial Monitor). The output is shown in the image below.



After several tests were conducted it had been found that it would just print a fixed number of "no motion" as well as a fixed number of "motion detected", a sort of nightmare loop. Unfortunately, it was unable to detect motion using the PIR sensor and was constantly printing out.

messages unnecessarily within the serial monitor. It could be said that led to switching the cables around like previously, in case their inverted. However, it was still not working and as a result, tried to change from G12 to G14 which unfortunately, still did not work.

At this point, it had been decided to conduct some further research in hope to find a replacement microcontroller, which could potentially be used to detect potential real-world threats. After conducting further research, another microcontroller had been discovered as the third and final attempt will reveal. (*For more screenshots, please refer to the Appendix.)*

4.5 Final Attempt: How to build a PIR motion detection sensor

The third and final attempt of building the PIR motion detection was successful. As mentioned previously, it was essential to conduct further research in hope to gain a new replacement microcontroller, which could potentially detect real-world threats. After further research had been conducted in the form of looking at reviews from experts, it could be said that the ELEGOO UNO R3 Board was discovered. As a result, the UNO R3 had been acquired and used to replace the previous ESP32 microcontroller development board. The required hardware for the final attempt is, as follows:

- ELEGOO UNO R3 Board ATmega328P ATMEGA16U2
- PIR Motion Sensor
- Jumper Cables Male to Female(MF) x3

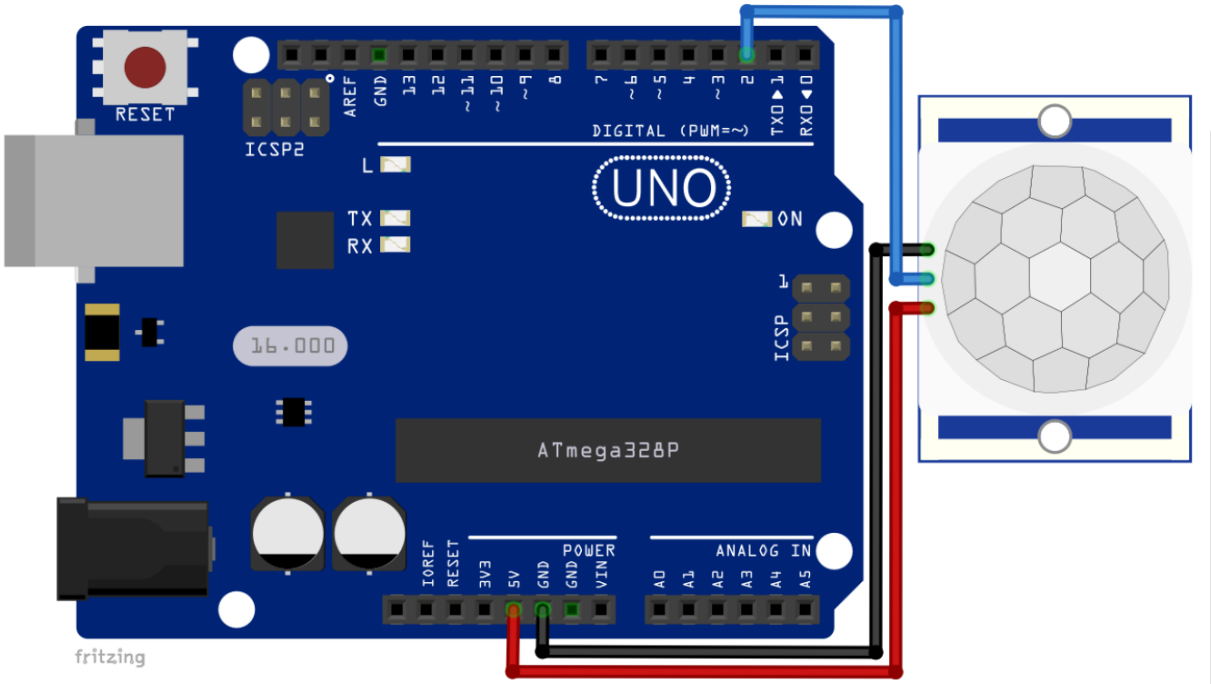
The diagram obtained taken from: Amazon.co.uk

The ELEGOO UNO R3 uses the same ATmega328 like the official Arduino UNO R3 board. The ATmega328 is the microchip on the board, which supports most of the microcontroller and peripheral features. It's also important to mention that microchips such as the ATmega328, are used in most IOT hardware such as televisions, GPS tracking etc. In this instance, the ATmega328 on the UNO R3 comes pre-programmed with a bootloader which allows users to upload new code to it, without the use of external hardware programmers. It could be said that the code which will be uploaded to the UNO R3, will be the code to detect potential real-world threats. In addition, the ATmega328 has 32 KB and has 2 KB of SRAM.

It is also possible to connect the board to a machine via USB, then updating the driver accordingly to the OS, in this case Windows 10. Furthermore, the Elegoo UNO R3 board is labelled on the side of the pin strips and as a result, it is easier to read when assembling the PIR motion sensor.

4.5.1 Connecting the hardware components

After learning about the Elegoo UNO R3, it could be said that it seems to have many improvements in comparison to the previous model used within the failed attempts. As a result, the Elegoo UNO R3 was later acquired however, it was essential to first revise the connectives again.

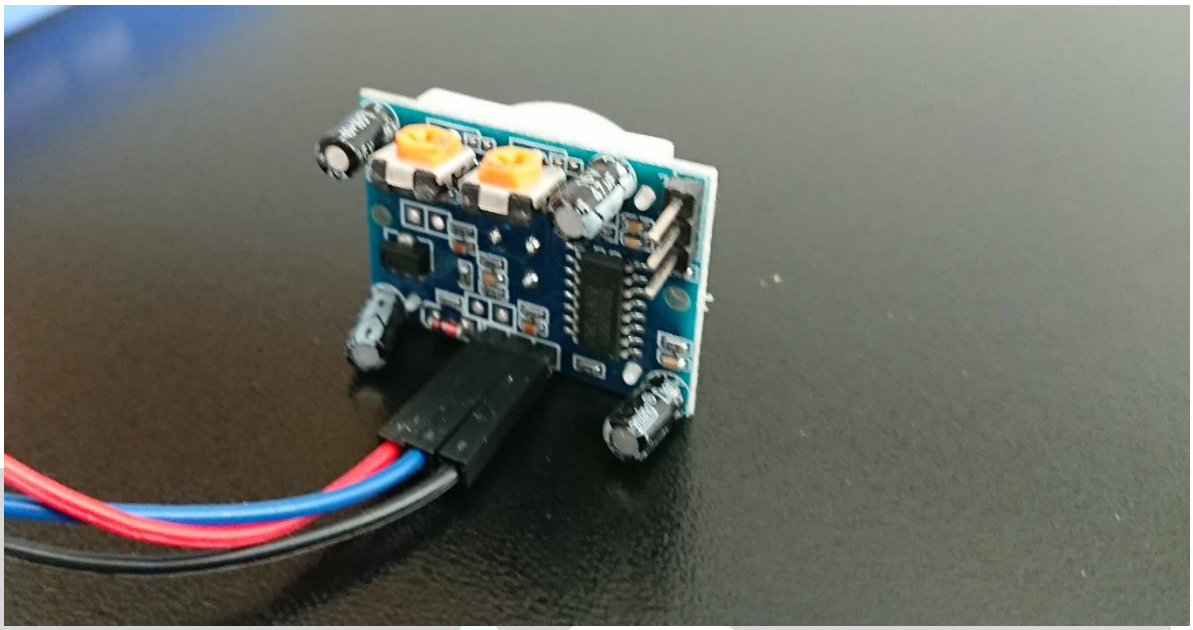


The diagram was obtained from: A-Z Delivery Whitepaper

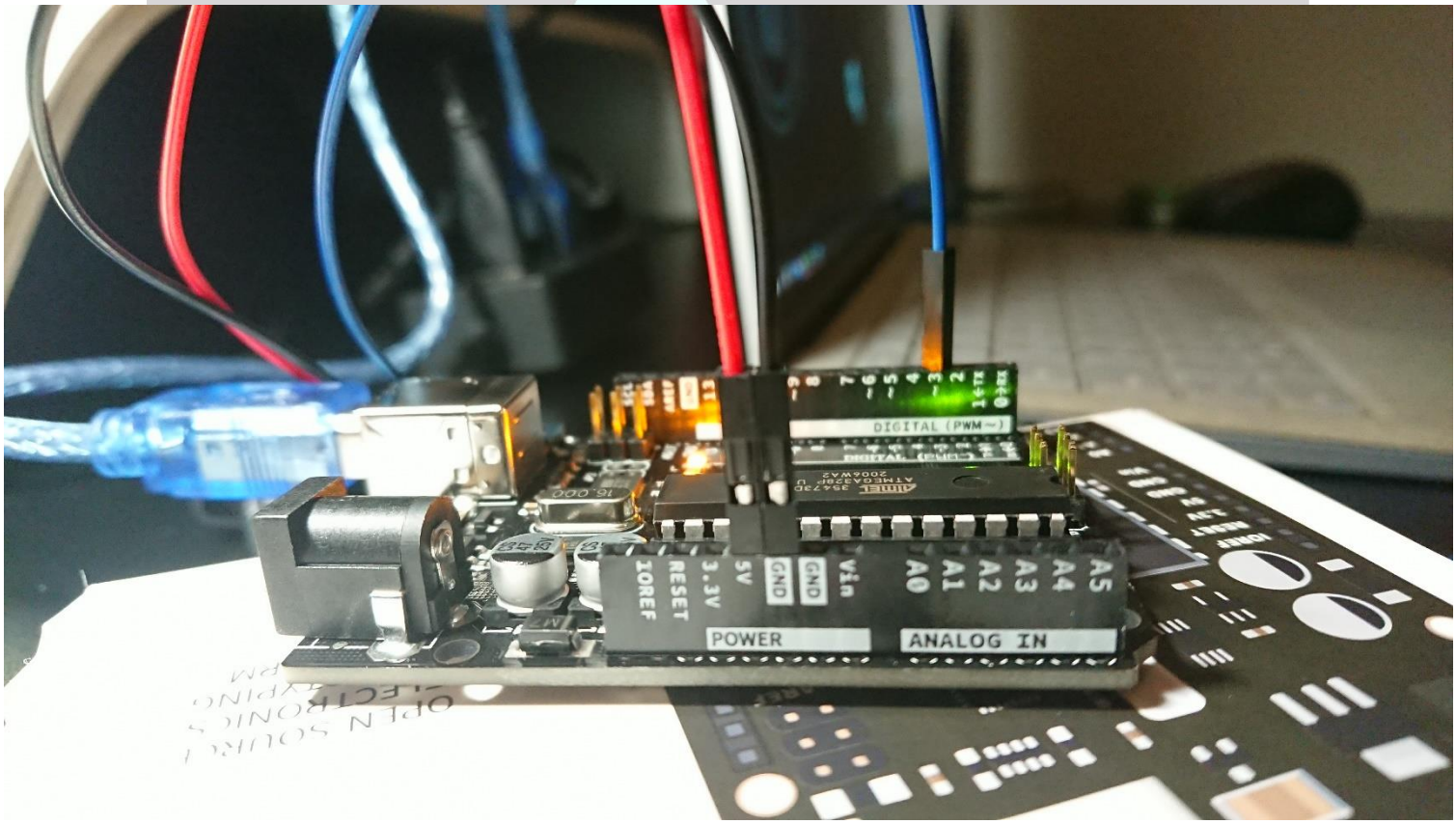
From looking at the diagram, it could be said that the PIR motion sensor needed to be connected accordingly, like the ESP32. Its clear the PIR VCC pin should be connected to the 5v Uno pin, then OUT pin into D2 and lastly, Ground Pin into the ground Uno pin.

PIR motion sensor PIN	Elegoo UNO R3	Wire colour
VCC	5V	RED WIRE
OUT	D2	BLUE WIRE
GND	GND	BLACK WIRE

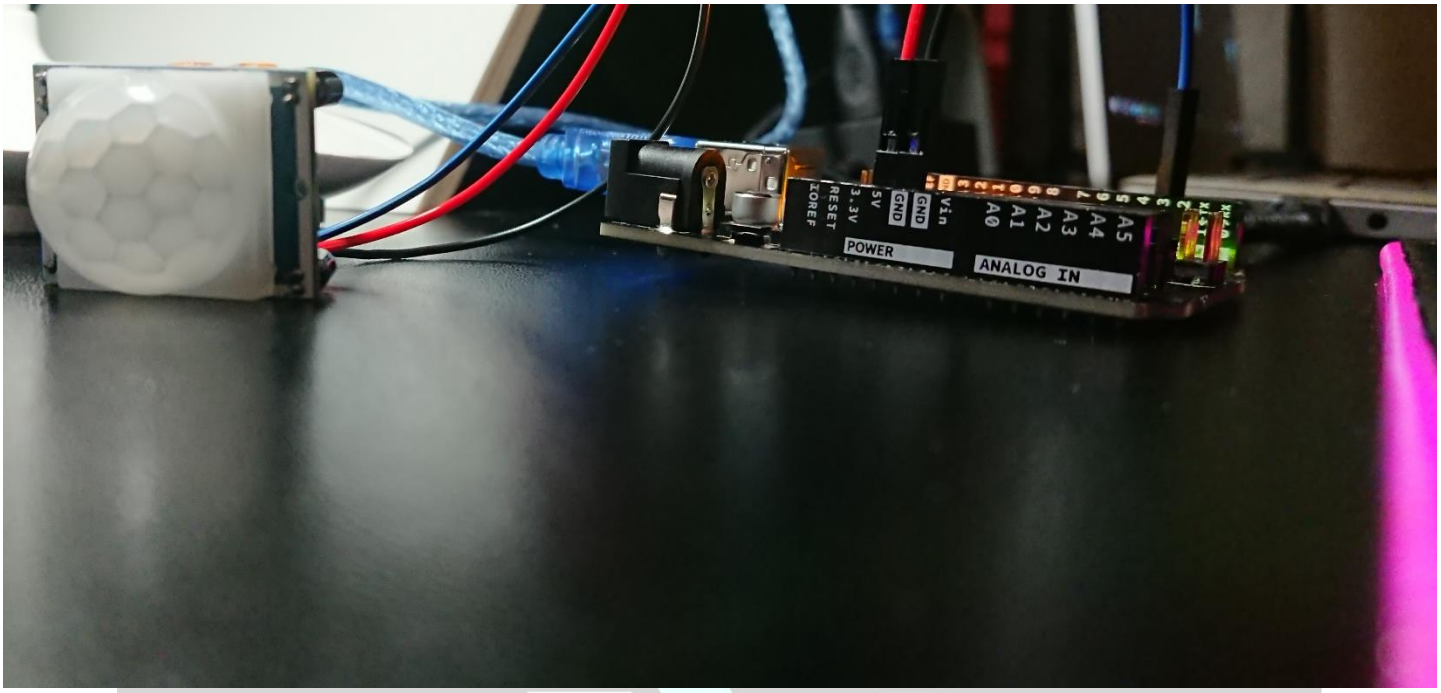
After the connectives had been revealed and clarified, it was essential the Elegoo UNO R3 was acquired and deployed as shown in the images below.



The image reveals the connectives to the PIR motion sensor from the Elegoo UNO R3.



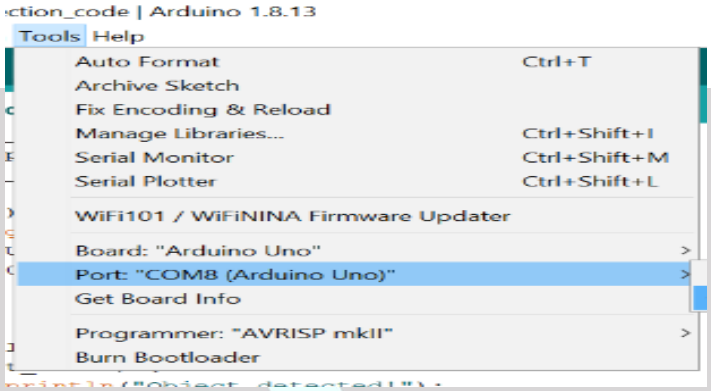
The image reveals the connectives to the Elegoo UNO R3 from the PIR motion sensor and the machine.



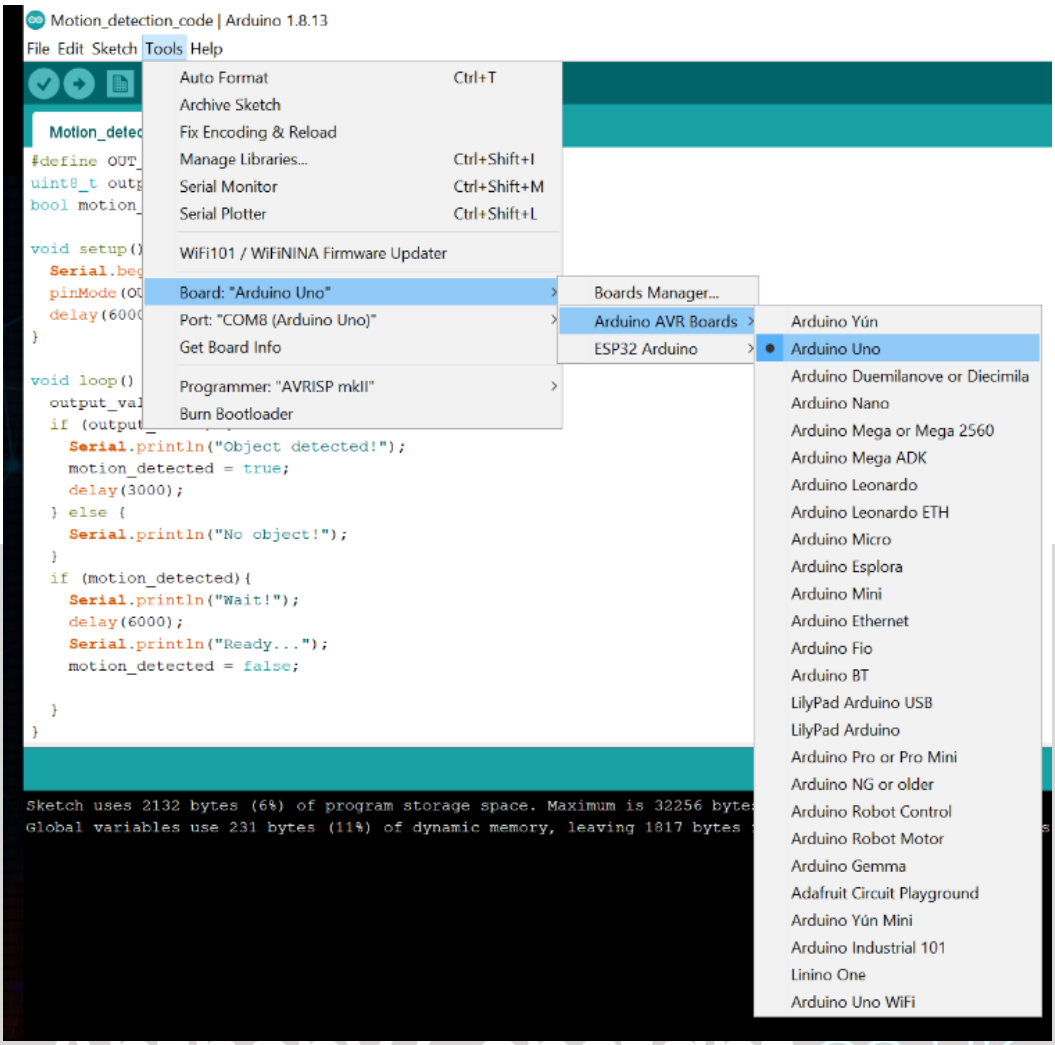
The image reveals all the connectives had been assembled.

4.5.2 Reconfiguring Arduino IDE settings from previous failed attempt

Whilst the PIR motion sensor was connected to the Elegoo UNO R3. It was required to connect the UNO R3 to the Windows 10 machine via USB and boot up Arduino IDE software, like the previous failed attempt. It could be said that it was decided to first implement enough code, to test whether the assembled artefact was working or not. However, the Arduino IDE had to be modified slightly from the previous attempt, as shown in the images below.



The image reveals choosing the Port the board is connected to from the machine.

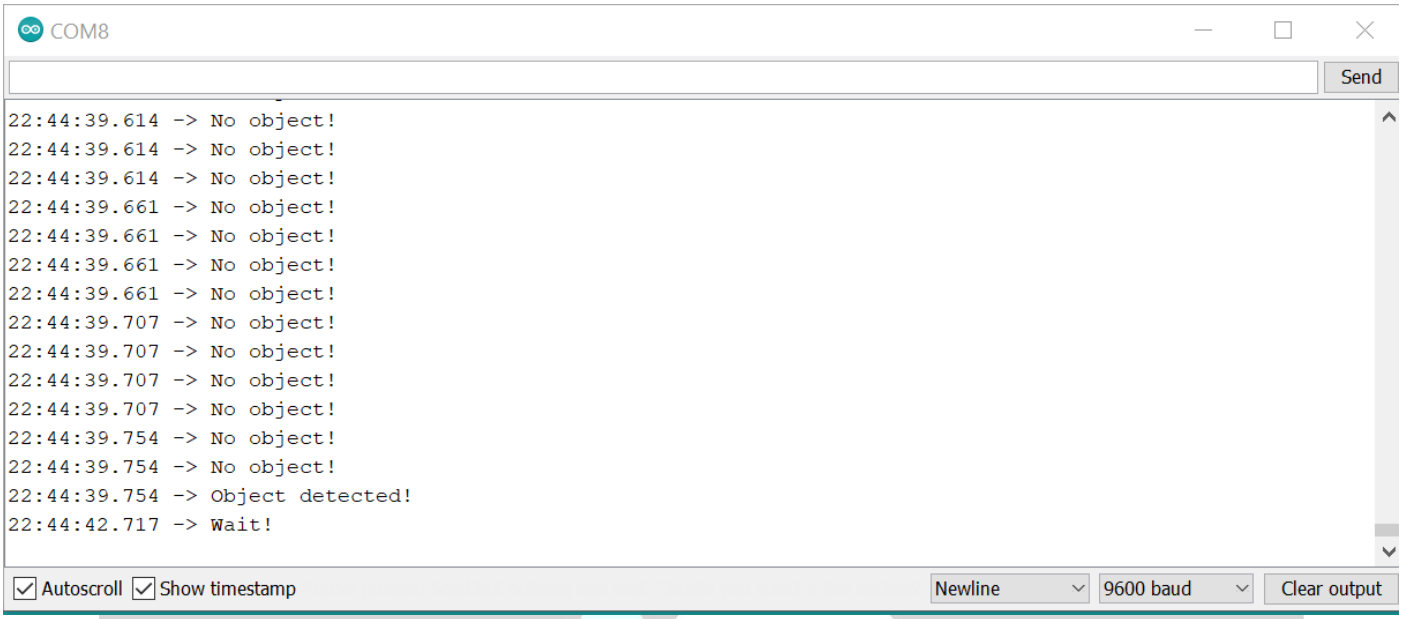


The image reveals choosing the specified board ‘Arduino Uno’

4.5.3 Testing: Refactoring the code in Arduino IDE to test the PIR motion detection sensor

The image reveals the test code which was used to test if the Artefact was successful or not. The code starts with defining which OUT_PIN from the PIR motion sensor is connected to on the UNO R3, in this case it was D2. Next, two variables called output_value and motion_detected are created. The variable output_value will store the state of the output(OUT) pin, whilst the motion_detected variable was used to display the output message. Within the setup() function the serial communication is started with a modulation rate of 9600. The OUT_PIN was set to INPUT and towards the end of setup() function, the delay was set to 60 seconds. Lastly, within the loop() function the state of the OUT_PIN is verified such that, if it is in HIGH state, the output message would be “Object detected!” and set to true. Otherwise, if the value of the OUT_PIN is in its LOW state, the output message would be “No Object!”. Whilst the value that’s stored within the

motion_detected variable is equal to true, the message output will be Wait! Which is then followed by a total of six seconds delay, before displaying “Ready…”. Besides making all the following changes



above, the rest of the settings had remained from the previous failed attempt. Next, was to upload the code to the UNO R3, by clicking “upload” within Arduino IDE. The code was successfully uploaded, taking up a total of 231 bytes (11%) of dynamic memory.

Whilst running the code within the Arduino IDE serial monitor, it could be said that the artefact was successfully working as shown in the image above. When no motion is detected in the real world, it constantly prints “No object!” then, when moving my hand in front of the PIR motion detection sensor, it prints “Object detected!”, until I move my hand away from the PIR motion sensor, it resets itself and prints “Wait!”. It was now evident the artefact could detect potential real-world threats. However, it had been decided to further develop the code, to alert the host in various ways once motion has been detected.

4.5.4 Further developing the motion detection sensor Code in Arduino IDE and Conclusion

Once the code had been further developed it was essential to compile the new code to check for any errors. Fortunately, there were no errors and the code were uploaded to the UNO R3 using a total of 237 bytes (11%) of dynamic memory, as shown in the image below.

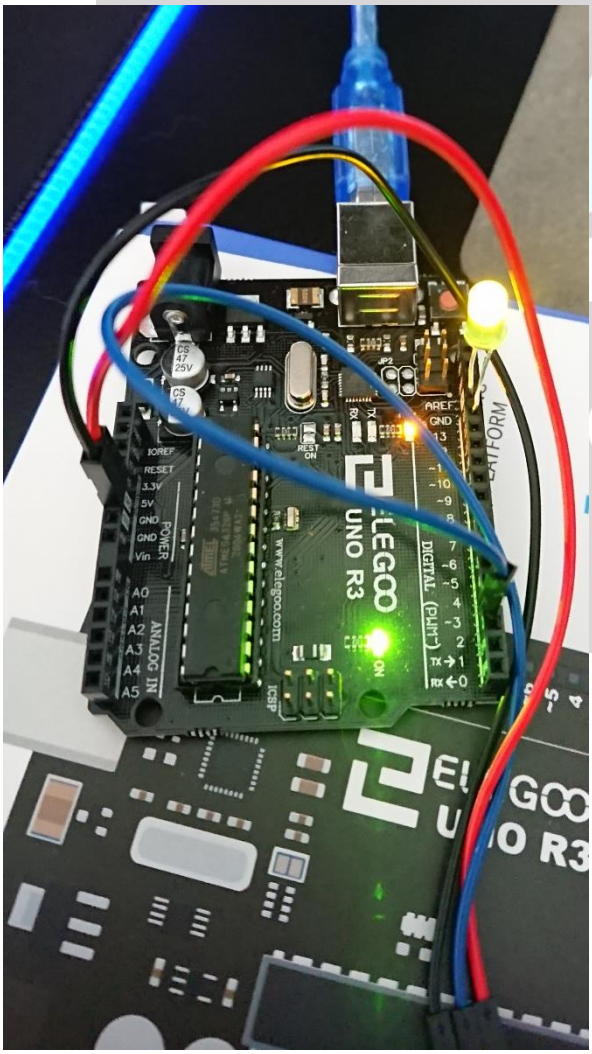
The image reveals the new code has been compiled successfully, without any errors. The following changes that were made within the code:

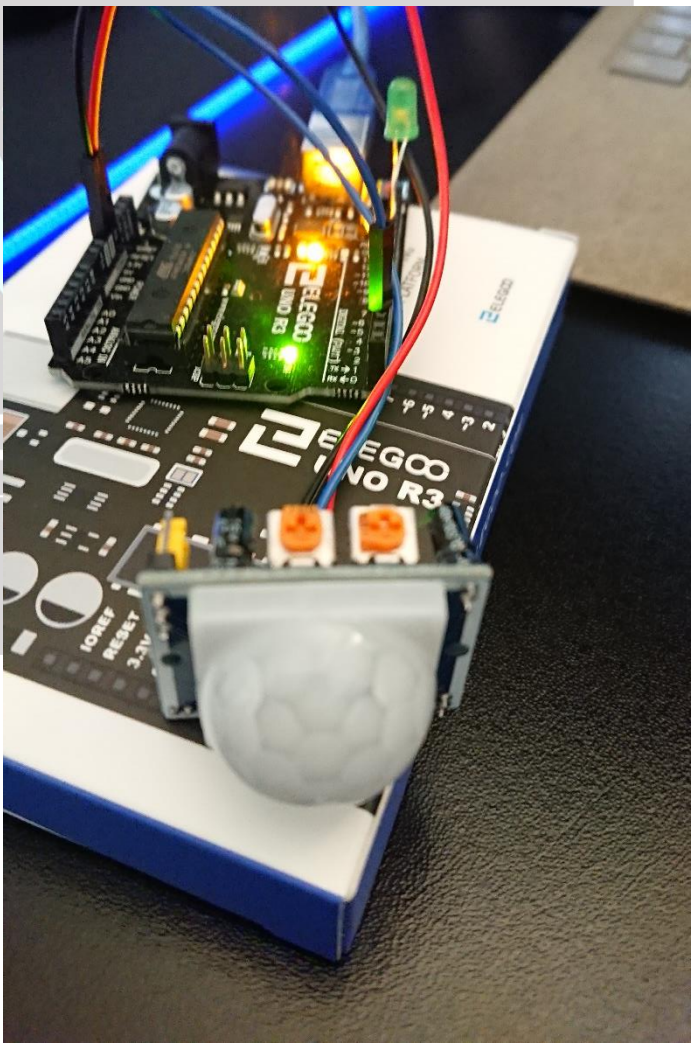
- Modified the OUT_PIN 2 to OUT_PIN 3

- Added LED functionalities ‘int led = 13’ Is defined according to the LED bulb, on the Uno R3 board. Furthermore, in void setup() the pinMode for the LED is set as output, as it is an output component. Within the void loop() IF statement, LED is set to high state if motion has been detected, which means the LED bulb should light up otherwise, it’s set to low state meaning the LED light will turn off again until motion has been detected.

- Modified the Syntax when motion is detected or not.

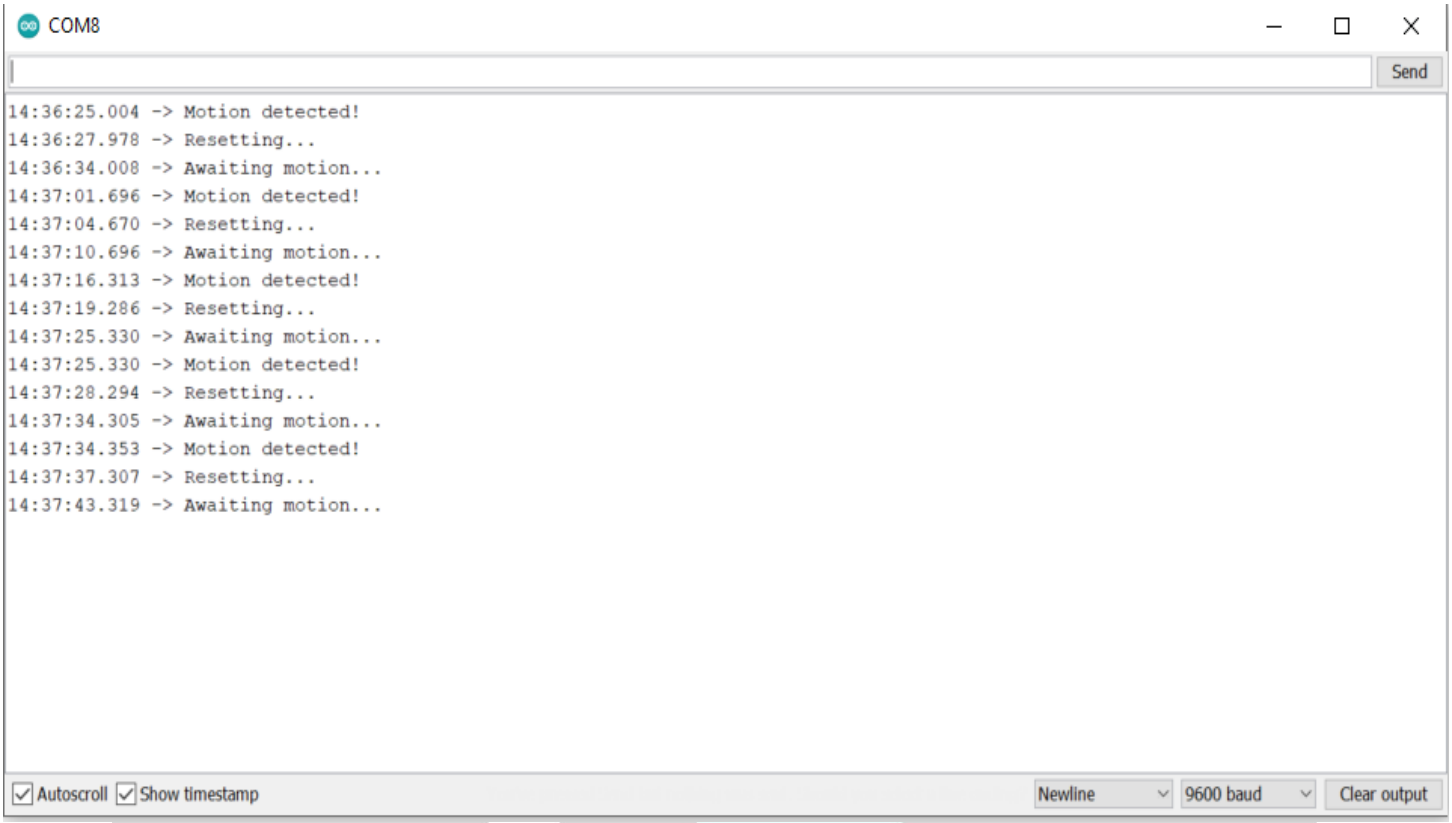
After making the following changes within the code, it was essential to add the physical LED bulb to the UNO R3 board as follows:





The images reveal the LED bulb’s short side is connected to the GND PIN and long side connected to PIN 13 on the UNO R3.

It also reveals the LED bulb lights up once motion has been detected otherwise, it turns off as shown in the images above.



The image reveals the Serial monitor, whilst PIR motion detection sensor is running. It can now be said the PIR motion detection sensor is able to detect potential real-world threats. With that said, the project report brief problem has been overcome.

5.0 Discussion and Evaluation

In this section it will be discussed and summarised the main findings including the results and evaluating what has been achieved and how I went about it. It is also important to discuss and evaluate the WireGaurd VPN protocol and the main findings from the installation up until running and maintaining the VPN server. This will clarify whether the initial brief problem of overcoming some internet privacy and security issues has been overcome. It's also important to discuss and compare the previous failed attempts in comparison to the final working artefact, regarding the PIR motion detection sensor. This will clarify whether it overcame the brief's secondary problem, of detecting potential real-world threats. Overall, the tasks that were carried out in this project report has provided good insight into Networking, Cyber-Security, Java programming and working with all the different acquired hardware components throughout this project report.

5.1 Chapter 1 Implementation

It could be said that at the beginning of the project report, it was essential to determine and evaluate the most suitable Virtual Private Network(VPN) protocols to implement, to overcome Internet privacy and security issues. However, it was essential to carry out a Literature review to gain a deeper understanding of the topics covered within VPNs.

My knowledge prior to undertaking the project report in Networking and Cyber-Security are considered intermediate, as I have studied these modules at the University of Hertfordshire and had further developed my knowledge. However, it could be said that I had little knowledge about OpenVPN and WireGaurd protocols, which meant the Literature review consisted of learning about these protocols, to ultimately choose which one to implement as part of the solution.

After I had conducted the literature review, it could be said that I had learned how both OpenVPN and WireGaurd operates in depth, and the key factors after comparing them, which divided them apart from each other. As a result, it was decided to choose WireGaurd as the main VPN protocol to implement as part of the solution. It could be said the main findings within the literature review consisted of learning about VPNs before learning about WireGaurd, to understand the fundamentals of how everything works. Such that, it was discovered VPNs were essentially an encrypted connection from an end device to a network. Also, Encryption and Decryption is a method of transforming readable data or plain text into a form that is unreadable, which is called cipher text. It was later discovered the differences between symmetric key and asymmetric public key cryptography. The key factors that were picked up were that Symmetric key encryption consists of one key (i.e. password), whereas Asymmetric key consisted of two keys, a public and private key. Furthermore, after combining the pieces of information previously discovered from the WireGaurd VPN protocol, it could be said that WireGaurd implements an Asymmetric public key encryption.

After conducting the research, it was then decided to find out how to implement a WireGaurd VPN server, to overcome some internet privacy and security issues. It was later found that to implement a WireGaurd VPN server, it would be more compatible to run a WireGaurd server on Linux OS and to use a Raspberry Pi 4 with PiVPN software installed. As a result, a Raspberry Pi 4 was deployed with PiVPN installed after I had conducted further research into the topic.

Throughout the process of implementing the WireGaurd VPN server, I had learned about the networking fundamentals such as, setting up a Static IP Address using Public DNS, working with Port numbers, then finally setting up the user clients to get everything connected. After setting up the VPN server, it was essential to test everything had worked and analyse the test results.

After investigating the configuration file that had been created from setting up the user clients on the VPN server, it had confirmed my findings from earlier that it was using Asymmetric public key encryption using various algorithms such as ChaCHa20 for symmetric encryption, BLAKE2s for hashing and keyed hashing, Curve25519 for ECDH and more according to the WireGaurd technical whitepaper.

The configuration file consisted of an Interface(VPN server) with the Private key stored and a Peer(user) which consists of the Public Key and a Pre-shared Key to access the server using the keys. Upon investigating the config file, it was essential to install the WireGaurd client at this point to conduct the testing phase. When the installation had been completed using my Windows machine, the config file had to be imported into the client to test the connectivity between server and client. The results had releveled that I had gained a successful connection which was very stabled between the server and client.

Overall, after analysing the test results it could be said that I had successfully set-up a remote access point to the WireGaurd VPN server and acquired a virtual IP address of **10.6.0.2**. Ultimately, by acquiring a virtual IP address it's able to mitigate the issues revolving around the privacy aspect of the project's entirety, by ensuring one remains unobserved or disturbed by authorities or anyone. However, whilst analysing the test results and from research it had been confirmed that WireGaurd stores the IP address of a client on the server. Although, this makes WireGaurd VPN run much faster than any other VPN, it poses limitations in terms of privacy. However, there are ways around this for example, NordVPN offer their own interpreted versions of WireGaurd protocol that work around the IP address problem, to implement a solution to the privacy issue. Whilst analysing the test results between server and client, it had been revealed it implements Asymmetric Key Encryption and therefore, eliminates the security aspect issues of the project's entirety, by preventing unauthorised users accessing the data being transmitted between the server and client.

5.2 Chapter 2 Implementation

For the second part of the project report it was essential to be able to detect potential real-world threats. Therefore, it was important to conduct research on what could potentially be implemented as part of the solution for this problem. It was later found that a PIR motion detection sensor could be used as part of the solution.

First, it was important to understand what a PIR motion sensor was, and how to implement one. As a result, I found that a PIR(Passive infra-red) are essentially used in security systems such as triggering an alert when motion has been detected. This was an essential piece of information as it inspired me to implement it as part of my own solution to detect potential real-world threats. For example, if a human walk past the PIR sensor's range of view, it could detect the small changes between the environment and human ultimately, alerting the host. With that said, it was important to find out how to implement a PIR motion detection sensor. As a result, I had conducted more research into the topic, and discovered an online video tutorial explaining in detail how to build it and included a list of the hardware components required.

After gathering all the required resources and learning about each hardware component then how they connect, it was essential to go ahead and build it. During the process of building the PIR motion sensor within the first attempt of chapter 2, it could be said that I gained a deeper understanding on working with the different hardware components, as I had no real prior knowledge, before undertaking the project. After adding all the connectives, it was essential to test the PIR motion sensor and analyse the test results. With that said, some test code was developed to test the status of the PIR motion sensor and as a result, I had learned more about programming in Python. The test consisted of running the test code within MU IDE, to check whether the PIR motion sensor can detect potential real-world threats using the code implemented. Unfortunately, after conducting the tests and analysing the results it could be said that the PIR motion sensor was unresponsive and not able to detect potential real-world threats such that, it led to further research in hope to find an alternative solution.

After the first attempt had failed and further research were carried out, the microcontroller was then implemented as part of the second attempt in chapter 2. This caused further delay within the project as additional hardware components were required as part of this solution such as a Breadboard and ESP32 Microcontroller. After these items were acquired it was important to learn about these hardware components in greater detail and expand on the PIR sensor to understand more in detail. After learning about the different hardware components, it was then essential to add the connectives. After rebuilding the PIR motion sensor using this method, the test code was refactored from the previous failed attempt within Arduino IDE after being properly configured. As a result, I learned in greater detail how Arduino IDE works with the ESP32 microcontroller

and PIR motion sensor which had further developed my Java programming skills. After the refactored code had been uploaded to the ESP32, it was necessary to analyse the test results to ensure it detects potential real-world threats. After several tests' attempts had been carried out under a various of circumstances, it was found that the PIR sensor seemed to be working but within its desired limitations such that, it picked up on the PIR sensor but not quite. Therefore, it was decided to implement the various tests conditions under different circumstances and tried to switch the jumper cables around in case the cables were inverted however, this was not the case. Therefore, it was decided to conduct further research in hope to acquire a replacement microcontroller which could detect potential real-world threats.

After the second attempt had failed and further research had been conducted, it could be said that the Elegoo UNO R3 microcontroller was discovered. After first learning in detail about the Elegoo UNO R3, it was essential to acquire it to replace the ESP32 microcontroller from the previous failed attempt. Fortunately, there was no further additional hardware components required other than the microcontroller itself, which did not cause much delay within the project. Once the Elegoo UNO R3 was acquired it could be said that it was essential to understand how to add the connectives correctly. As a result, I had learned in more detail how hardware components generally connect to another component using jumper cables. After learning and building the PIR motion senor it was essential to reconfigure the Arduino IDE settings from the previous failed attempt for example, switching the port and board settings accordingly. I had learned how to reconfigure the settings within Arduino IDE according to my own device. It was essential to refactor the code in Arduino IDE from the previous failed attempt, to test if the PIR motion sensor can detect potential real-world threats. Throughout the process of refactoring the code, I had learned more about Java programming and had extended my current knowledge.

After uploading the refactored code to the Elegoo UNO R3 it was essential carry out the tests and analyse the results. After several tests were carried out under various circumstances it could be said that it was successful and able to detect potential real-world threats. The testing phase consisted of using the test code within Arduino IDE, to check whether it could detect potential real-world threats. More tests were carried out to ensure the artefact can detect potential threats by checking whether it works in a dark room, which had also been successful. However, it was essential to further develop the motion detection sensor code in Arduino IDE to ensure it could detect potential real-world threats and alert the host in various ways with precision. As a result, I had learned in more detail about my working artefact and Java programming whilst developing the code.

5.3 Project Aims and Objectives

The primary project aim was to eliminate some Internet privacy and security issues and I believe it has been accomplished with certain limitations. Such that, it successfully eliminates privacy issues by providing a virtual IP Address and creates a remote access point using a tunnel, where unauthorised users are unable access it unless they have the key(config file). Unfortunately, privacy has not been eliminated entirely at this point as the WireGaurd VPN server is currently pre-configured to store the user's IP address on the server. I also believe the issues revolving around the security aspects has been mitigated completely, as the WireGaurd VPN implements Asymmetric public key encryption, preventing unauthorised users accessing the data between server and client.

What I have learned from this whole experience is how to implement and maintain a WireGaurd VPN server. In conclusion I believe I have further developed my current knowledge regarding Networking and Cyber-Security as an entirety. In the future I hope to reconfigure the WireGaurd protocol to disallow user's IP addresses to be stored on the server and ultimately, mitigating the privacy issues completely.

The secondary project aim was to develop a system that could detect potential real-world threats. I believe I have found the solution to this problem, by using a PIR motion sensor with a microcontroller and Java programming code in Arduino IDE to control the behaviour of when motion is detected. It includes the timestamp of when motion had been detected and the LED bulb will light up as an indication, motion has been detected.

What I have learned from this whole experience is how to build a PIR motion detection sensor to detect potential real-world threats. In conclusion, I believe I have further developed my knowledge in working with hardware components, and I am now able to better analyse problems carefully and sufficiently finding the solutions to the problems. Therefore, I understand the importance of being able to adapt under various circumstances. In the future I hope to extend my PIR motion detection sensor code, to perhaps alert the host via Email, when motion has been detected which will ultimately, alert the host even when away from the desk.

References

Cisco (2020) What is a VPN? – Virtual Private Network. Available at: https://www.cisco.com/c/en_uk/products/security/vpn-endpoint-security-clients/what-is-vpn.html#~how-vpn-works [Accessed 28th January 2021]

Paul Bischoff (2019) VPN protocols explained and compared. Available at: https://www.comparitech.com/vpn/protocols/ [Accessed 28th January 2021]

Tim Mocan (2019) What Is OpenVPN & How Does OpenVPN Work? Available at: https://www.cactusvpn.com/beginners-guide-to-vpn/what-is-openvpn/ [Accessed 04th February 2021]

Ajay Ohri (2021) Symmetric and Asymmetric Key Cryptography: All You Need To Know in 3 Points. Available at: https://www.jigsawacademy.com/blogs/cyber-security/symmetric-and-asymmetric-key-cryptography/#:~:text=The%20basic%20difference%20between%20symmetric,uses%20another%20key%20for%20decryption.&text=It%20is%20also%20known%20as%20public%2Dkey%20cryptography. [Accessed 04th February 2021]

OpenVPN (2021) The History of OpenVPN. Available at: https://openvpn.net/the-history-of-openvpn/ [Accessed 09th February 2021]

AbhayBhat (2019) Blowfish Algorithm with Examples. Available at: https://www.geeksforgeeks.org/blowfish-algorithm-with-examples/ [Accessed 09th February 2021]

OpenVPN (2021) Why does OpenVPN use UDP and TCP. Available at: https://openvpn.net/faq/why-does-openvpn-use-udp-and-tcp/#:~:text=The%20OpenVPN%20protocol%20itself%20functions,that%20fails%2C%20then%20try%20TCP. [Accessed 25th February 2021]

Sven Taylor (2020) WireGaurd VPN: What You Need to Know. Available at: https://restoreprivacy.com/vpn/wireguard/ [Accessed 25th February 2021]

Heinrich Long (2021) WireGaurd vs OpenVPN. Available at: https://restoreprivacy.com/vpn/wireguard-vs-openvpn/#:~:text=WireGuard%20also%20differs%20from%20OpenVPN,an%20additional%20layer%20of%20security. [Accessed 25th February 2021]

Emmet (2020) Setting up a WireGaurd VPN on the Raspberry Pi. Available at: https://pimylifeup.com/raspberry-pi-wireguard/ [Accessed 25th February 2021]

Laura (2019) WHAT ARE PIR ALARMS AND HOW DO THEY WORK? Available at: https://www.bulldog-securityltd.co.uk/what-are-pir-alarms-and-how-do-they-work/#:~:text=PIR%20stands%20for%20passive%20infra,amount%20of%20this%20infrared%20activity. [Accessed 28th February 2021]

Tech With Tim (2019) How to Use a PIR Motion Sensor with the Raspberry Pi Available at: https://www.youtube.com/watch?app=desktop&v=Tw0mG4YtsZk&t=40s&ab_channel=TechWithTim [Accessed 30th February 2021]

Raspberry Pi (2021) Raspberry Pi OS Available at: https://www.raspberrypi.org/software/ [Accessed 03rd March 2021]

PiVPN (2021) PiVPN Available at: https://www.pivpn.io/ [Accessed 03rd March 2021]

AZ-Delivery (2021) AZ-Delivery Available at: https://www.az-delivery.de/en/ [Accessed 08th March 2021]

A-Z Delivery (2021) ESP-32 NodeMCU Development board Pinout Diagram. Available at: https://cdn.shopify.com/s/files/1/1509/1638/files/ESP-32_NodeMCU_Developmentboard_Pinout.pdf?v=1609851295 [Accessed 08th March 2021]

A-Z Delivery (2021) Motion sensor Motion detection module HC-5R501 PIR for Arduino. Available at: https://www.az-delivery.de/en/products/bewegungsmelde-modul[Accessed 08th March 2021]

A-Z Delivery (2021) Microcontroller Board ATmega 328. Available at: https://cdn.shopify.com/s/files/1/1509/1638/files/Mikrocontroller_ATMega328_16U2.pdf?v=1602863871 [Accessed 08th March 2021]

Amazon (2021) ELEGOO UNO R3 Board ATmega 328P. Available at: https://www.amazon.co.uk/ELEGOO-Board-ATmega328P-ATMEGA16U2-Cable/dp/B01EWOE0UU/ref=sr_1_4?dchild=1&keywords=elegoo+uno+r3&qid=1616759142&sr=8-4 [Accessed 08th March 2021]

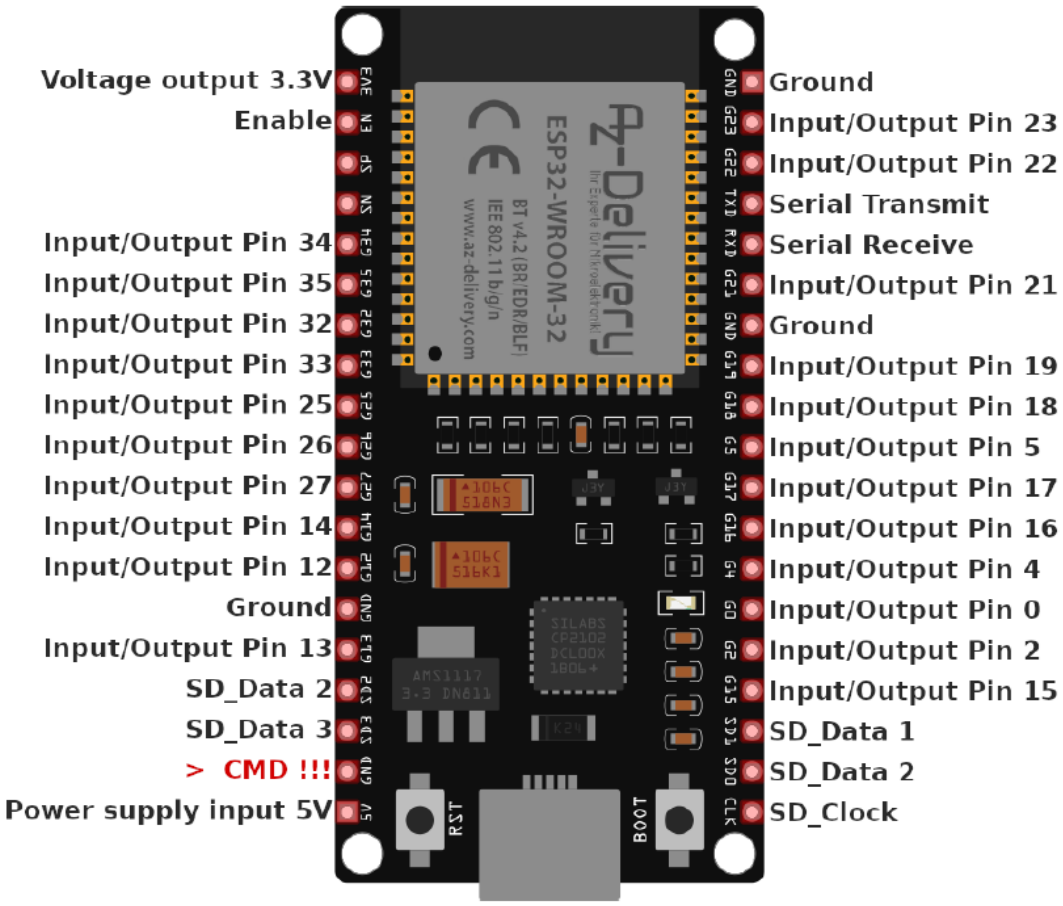
Appendix A

Specifications of the ESP32 NodeMCU

Power supply voltage (USB)	5V
Input/Output voltage	3.3V
SoC	ESP32-WROOM-32
CPU	Xtensa 32-bit LX6 microprocessor
Clock frequency	up to 240MHz
Instruction RAM	512kB
External flash memory	up to 4MB
GPIO digital pins	16 (can be configured as PWM)
Analog to digital pins	18
Analog to digital resolution	12bit
Digital to analog pins	2
Digital to analog resolution	8bit
UART	3
I2C interfaces	2
I2S interfaces	2
SPI interfaces	4
Touch-sensor inputs	10
IR remote controller pins	8
Integrated Hall-effect sensor	
Ultra low-power analog preamp	
Multiple real-time clock	
USB serial chip	CP2102
PCB antenna	
Output antenna power	+ 19.5dBm in 802.11b mode
Wi-Fi	2.4 GHz up to 150 Mbits/s
Wireless protocols	802.11 b/g/n
Bluetooth	BLE and legacy Bluetooth
Integrated TCP/IP Protocol Stack	

Pinout of the ESP32 NodeMCU

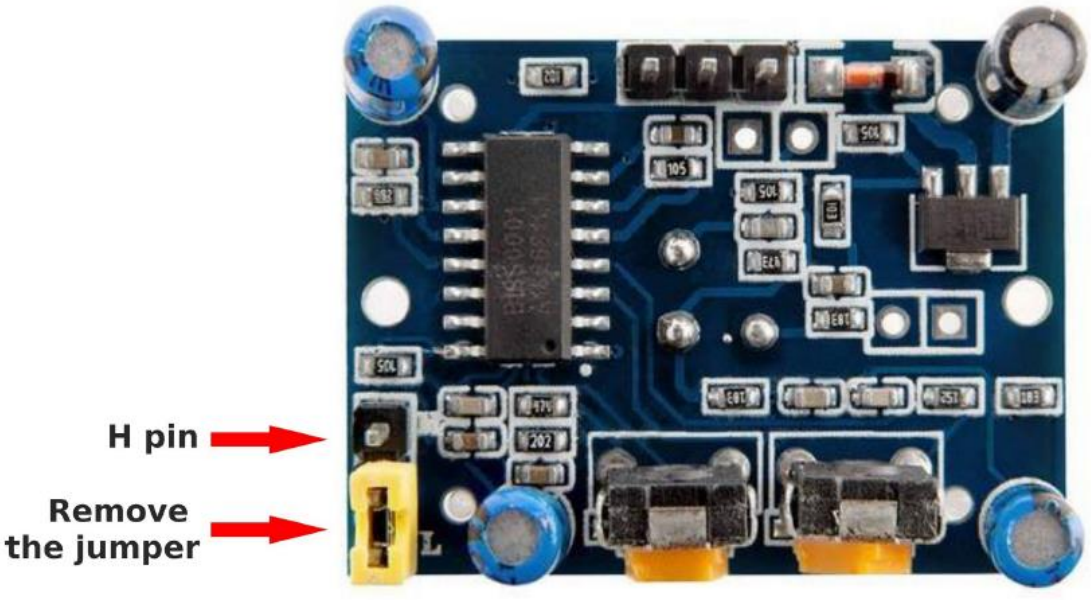
The ESP32 NodeMCU has 38 pins. Pinout diagram is shown on the following image:



Specifications of the PIR Sensor

Power supply voltage:	5V DC
Operational temperature:	from -15 to +70℃
Lock time:	200 milliseconds
Sensing :	110 degrees
Distance range:	up to 7 meters
Block time:	from 2.5sec. (default) up to 10 sec.
Trigger methods:	L - disable, H - enable repeat trigger
TTL output:	LOW – 0V, HIGH – 3.3V (logic levels)
Quiescent current:	less than 50μA
Power consumption:	65mA
Lens diameter:	23mm
Dimensions:	33 x 25 x 30mm [1.3 x 1 x 1.2in]

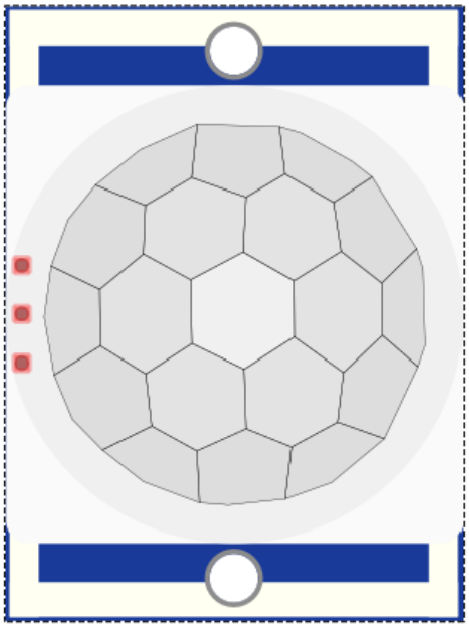
Note: Power supply has to be as stabe as possible or else there would be errors in the sensor output caused due unstable power supply.



The pinout

The HC-SR501 PIR module has a three pins. The pinout is shown on the following image:

Ground - GND
Digital output - OUT
Power supply +5V - VCC



Note: Prefered operating power supply voltage range is *5V*. The output voltage on the digital pin is in *3.3V* range.

Note: When the module is connected to the power supply, the sensor takes from *30* up to *60* seconds to warm up, stabilize.


File Edit Tabs Help
i@raspberrypi:~ $ pivpn -qr wgadmin
:: Showing client wgadmin below
i@raspberrypi:~ $