

Zero Trust IoT implementation and evaluation on the importance of zero trust structure in organizations for IoT systems

Benjamin Lewis Alpha Saxon

SID: 2108502

MOD002691 Final Project

BSc (Hons) Cyber Security

Submitted: April 2024

# Abstract

The exponential growth in the adoption of Internet of Things (IoT) devices has allowed organisations to access unprecedented connectivity providing data-driven insights, however, security implementations commonly fall short of protecting data, and systems and supporting innovation. With the use of successful zero trust implementation, the time to deploy a well-protected system can be significantly reduced while also increasing the security posture of all an organizations IoT devices as well as other devices such as servers and computers.

This report will look at how to implement a secure IoT-powered platform while allowing remote connections for both IoT devices and users through zero trust best practices such as access control, encryption in transit, micro-segmentation, and preventative measures such as not needing to expose endpoints to public-facing networks using VPN tunnelling.

The results of the report concluded that by implementing systems that follow zero trust principles data security and integrity can be ensured to a high degree. The results also show the importance of implementing access control and segmenting systems where possible as a way to reduce the impact in the event of a breach of one system to prevent movement to other systems.

# Acknowledgements

I would like to thank my Father and Grandparents Matthew S, Rodny A, and Jennifer A, for supporting me through my time at university for their unwavering support throughout my journey at university. Their encouragement, guidance, and love have been invaluable to me, as well as my closest friends Tom P, and Lewis M for being there for me during my time at university when physical distance didn’t prevent them from continuing to support me through their friendship along with my fellow student Matt B who kept me on track through my assignments.

Contents

[Abstract ii](#_Toc163839163)

[Acknowledgements iii](#_Toc163839164)

[1 Introduction viii](#_Toc163839165)

[1.1 Aims of Project viii](#_Toc163839166)

[1.2 Background viii](#_Toc163839167)

[2 Literature review x](#_Toc163839168)

[2.1 Overview x](#_Toc163839169)

[2.2 Aim of literature review x](#_Toc163839170)

[2.3 Academic paper 1 - NIST Special Publication 800-207 x](#_Toc163839171)

[2.4 Academic paper 2 - BlueSky: Towards Convergence of Zero Trust Principles and Score-Based Authorization for IoT Enabled Smart Systems xii](#_Toc163839172)

[2.5 Academic paper 3 - Access Control Enforcement in IoT: state of the art and open challenges in the Zero Trust era xiii](#_Toc163839173)

[2.6 Literature Review reflection xiv](#_Toc163839174)

[3 Methodology xv](#_Toc163839175)

[3.1 Methodology of Implementation xv](#_Toc163839176)

[3.1.1 Frontend Implementation Approach with Next.js xix](#_Toc163839177)

[3.1.2 Backend Implementation Approach with Node.js and WebSocket handling xix](#_Toc163839178)

[3.1.3 0Auth and JWT’s xx](#_Toc163839179)

[3.1.4 Implementation Approach for Tailscale, ACL’s and MagicDNS xxiii](#_Toc163839180)

[4 Implementation xxv](#_Toc163839181)

[4.1 Network topology xxv](#_Toc163839182)

[4.1.1 Network level access control (NAC) xxvi](#_Toc163839183)

[4.1.2 Tailscale, Tailnet implementation xxvi](#_Toc163839184)

[4.2 Code Implementations xxviii](#_Toc163839185)

[4.2.1 Software level access control xxviii](#_Toc163839186)

[4.2.2 Dashboard Development xxx](#_Toc163839187)

[4.2.3 IoT device communication xxxi](#_Toc163839188)

[4.3 Further implementations to Practises and Processes within an organisation for ZTA. xxxv](#_Toc163839189)

[4.3.1 Continuous Monitoring and Risk Assessment xxxv](#_Toc163839190)

[4.3.2 Dynamic Access Control (DAC) xxxvi](#_Toc163839191)

[4.3.3 Micro-Segmentation xxxvii](#_Toc163839192)

[4.3.4 Identity-Centric Security xxxviii](#_Toc163839193)

[4.3.5 Compliance and Regulatory Requirements xl](#_Toc163839194)

[5 Results xli](#_Toc163839195)

[5.1 Summary of Implementation xli](#_Toc163839196)

[5.1.1 IoT Client xlii](#_Toc163839197)

[5.1.2 Server-Side Functionality overview and results xlviii](#_Toc163839198)

[6 Discussion l](#_Toc163839199)

[6.1 Evaluation of Zero Trust implementation l](#_Toc163839200)

[6.2 Comparison with Traditional Security Models li](#_Toc163839201)

[7 Conclusion liii](#_Toc163839202)

[7.1 Summary and evaluation of research and implementation. liii](#_Toc163839203)

[7.2 Future work and Difficulties liv](#_Toc163839204)

[8 References lv](#_Toc163839205)

[Appendix I – Application setup lix](#_Toc163839206)

[Preconfigured Virtual Machine setup: lix](#_Toc163839207)

[Self-setup: lx](#_Toc163839208)

[Appendix II – Main code and code referred in main text lxii](#_Toc163839209)

[Monitor.js code: lxii](#_Toc163839210)

[pingServer.js code: lxix](#_Toc163839211)

[App.js code: lxx](#_Toc163839212)

[page.tsx code: lxxi](#_Toc163839213)

[add-card.js code: lxxvi](#_Toc163839214)

[ipMac.js code: lxxxiv](#_Toc163839215)

[pingUtility.js code: lxxxvii](#_Toc163839216)

[profile.js code: lxxxvii](#_Toc163839217)

[startservers.sh code: xc](#_Toc163839218)

[client.py – IoT client code: xci](#_Toc163839219)

[IoT client config.json code: xcv](#_Toc163839220)

[Tailscale Access Control List: xcvi](#_Toc163839221)

List of Figures

[Figure 1- Encoded JWT Token xx](#_Toc163434240)

[Figure 2 - Decoded JWT Header xxi](#_Toc163434241)

[Figure 3 - Decoded JWT Payload xxi](#_Toc163434242)

[Figure 4 - Decoded JWT Signature xxii](#_Toc163434243)

[Figure 5 - JWT request process xxiii](#_Toc163434244)

[Figure 6 - JWT verification and authentication process xxiii](#_Toc163434245)

[Figure 7 - Network typology xxv](#_Toc163434246)

[Figure 8 - Assigned IP addresses in the CGNAT range xxvi](#_Toc163434247)

[Figure 9 - ACL xxvii](#_Toc163434248)

[Figure 10 - 0Auth login provider xxviii](#_Toc163434249)

[Figure 11 - Fetch client access token xxix](#_Toc163434250)

[Figure 12 - App.js Domain & Client ID xxix](#_Toc163434251)

[Figure 13 - 0Auth Permissions Dashboard xxix](#_Toc163434252)

[Figure 14 - Information Card layout xxxi](#_Toc163434253)

[Figure 15 - Device status indicators xxxi](#_Toc163434254)

[Figure 16 - API to Websocket interaction xxxii](#_Toc163434255)

[Figure 18 - Micro-Segmented network xxxviii](#_Toc163434256)

[Figure 19 - Client initiates TLS handshake xlii](#_Toc163434257)

[Figure 20 - Server response to TLS handshake (Server Hello, Key exchange, Cipher Spec) xlii](#_Toc163434258)

[Figure 21 - Detailed Server response to TLS handshake (Server Hello, Key exchange, Cipher Spec) xliii](#_Toc163434259)

[Figure 22 - Response to sendinfo command xlvi](#_Toc163434260)

[Figure 23 - Response to device\_prop command xlvi](#_Toc163434261)

[Figure 24 - Auto-Reconnection function output xlvii](#_Toc163434262)

List of Tables

[Table 1 - Project metrics xv](#_Toc163434263)

[Table 2 - ACL breakdown xxvii](#_Toc163434264)

[Table 3 - Limited command-based structure table: xxxiii](#_Toc163434265)

[Table 3 - Project outcome tests xli](#_Toc163434266)

[Table 4 - Server functionality providers xlviii](#_Toc163434267)

[Table 5 - Security Model pros and cons li](#_Toc163434268)

# 1 Introduction

## Aims of Project

Cyber security is a high priority when implementing technologies such as IoT devices into an organisation's network and becomes increasingly challenging when implementing devices at remote locations or outside of an organisation's normal network. This project aims to demonstrate how using zero-trust methodologies can aid a secure implementation regardless of device location along with the benefits of a zero-trust approach over historical approaches such as ring-fencing and Castle-and-Moat Models.

* Evaluate the benefits of zero trust over traditional approaches to security.
* Evaluate the benefits and shortcomings of traditional approaches to security.
* Design a web application that allows secure access to data from IoT devices.
* Implement a modular code base allowing for scalability of services.

## 1.2 Background

Recent changes in the way organisations approach the utilisation of information systems such as remote working and IoT devices have left a gap in security approaches due to the outdated threat model the security architecture was built around. IoT-focused attacks increased by 373.797% from 2018 to 2022 as shown in the research undertaken by (Petrosyan, 2023) and with a prediction of 29.42 billion devices to be connected to the internet by 2030 according to the research of (Vailshery, 2023), security architecture needs rapid improvement. Zero-Trust architecture implements mitigations for all threat actors due to a holistic approach founded around the concept of least privilege.

Zero-trust architecture is built around the principle that threats exist inside and outside of a network perimeter therefore restricting access to the lowest level needed for a user or service to complete tasks as well as taking precautionary mitigating measures such as micro-segmentation of network sub-sections and services, comprehensive monitoring measures and robust data encryption to detect, contain and minimise breaches in security.

Considering the challenges and threats currently facing IoT-based cyber security, zero trust architecture creates a strong ground for cementing the foundations of a rapidly growing industry which requires the need for fast implementation while remaining secure which is only further reinforced by the study commissioned by Microsoft and conducted by (FORRESTER, 2021) which stated moving to a Zero Trust Solution, “Reduced effort required to provision and secure new infrastructure by 80%.”, the study also found that in the case of Microsoft, the risk of a data breach was reduced by “50%” through the core components of zero trust solutions which are, improved authentication, network, and endpoint security protocols.

# 2 Literature review

## 2.1 Overview

Security architecture evolves over time as the landscape of threats and requirements of a system change. There are many different solutions to security risks that have been developed and different viewpoints on which approach provides the best defence against cyber threats. The use of past papers will aid in providing a fair view of which method best supports modern requirements.

## 2.2 Aim of literature review

The key aim of completing the literature review is to review past work to evaluate the concepts, principles, and effectiveness of zero-trust threat models and alternative threat models in relation to modern security requirements and threats.

## 2.3 Academic paper 1 - NIST Special Publication 800-207

The research conducted by (Rose et al., 2020) evaluates ZTA (Zero Trust Architecture) and challenges traditional perimeter-based security approaches. NIST Special Publication 800-207 outlines how modern operational practices of organisations utilising dated security models are increasingly vulnerable to internal attacks due to factors such as BYOD (Bring Your Own Device) and remote working implementation which allow remote access to a network and the resources within that network, due to a lack of internal controls such as access control and micro-segmentation of services.

The publication outlines 7 basic tenets of ZTA through a “technology agnostic” approach and highlights the difficulties of applying policies to external users and public-facing applications. The publication also highlights issues that can’t be mitigated by historic security architectures such as cloud computing due to resources being outside of an organisation’s network perimeter along with devices inside of an organisation’s network that can’t be trusted due to a lack of ownership or configuration ability defined as nonenterprise-owned assets including BYOD.

One of the 7 tenets of ZTA the publication investigates is dynamic access control which monitors several metrics to determine the security posture of a device as well as observed behaviour and environmental attributes referred to as client identity to reduce the risk of breaches via compromised devices. Dynamic access control allows for policies and access levels to change in real-time to ensure a secure and adaptive environment capable of protecting organisation assets and resources from external and internal threats while ensuring availability where requirements are met.

The research provided by (Rose et al., 2020) is beneficial to the development of the methodology and implementation of the project as it has highlighted how zero trust resolves issues with historic approaches to security and provided valuable insights into technologies not planned to be used in this project but could be included if expanded from an application implementation to a network implementation.

## 2.4 Academic paper 2 - BlueSky: Towards Convergence of Zero Trust Principles and Score-Based Authorization for IoT Enabled Smart Systems

The second paper researched is the work of (Ameer et al., 2022) which outlines the challenges faced in IoT access control models and proposes a specification for authorization models tailored to IoT systems which are crucial for designing access control methods.

The paper also highlights the necessity of zero trust tenets and their importance relating to IoT systems such as dynamic access control, fine-grained authorisation, the issues of limited computational power associated with IoT devices, the lack of a compliance standard relating to IoT devices and the difficulties of scaling security measures for IoT devices due to the quantity of devices. The research highlights many different issues around IoT security and considers more than just security policy and procedures by also examining the issues of security when computational resources are low compared to traditional resource levels in servers and workstations combined with the acknowledgement that IoT systems are commonly implemented over large geographical areas.

Central to the paper by (Ameer et al., 2022) is the development of a structured methodology for providing a structured approach to implementing access control in zero trust systems focused on the tenets of zero trust researched in paper 1 and the PEI (Policy, Enforcement, and Implementation) model.

The information provided in the paper produced by (Ameer et al., 2022) has provided great insights into the problems and solutions of IoT security when systems are considered for remote and large-scale implementations as it provides a valuable consideration for both the selection of hardware to use and the consumption of resources needed to implement a high level of security such as a zero-trust model.

## 2.5 Academic paper 3 - Access Control Enforcement in IoT: state of the art and open challenges in the Zero Trust era

The third paper researched is the work of (Colombo, Ferrari and Tümer, 2021) which focuses on different ways to implement access control in relation to IoT devices and the policies that should be implemented within access control such as least privilege access. The paper outlines many mechanisms that could be used for access control such as role-based, attribute-based, capability-based, and usage-based access control and evaluate the benefits from both a security standpoint and a resource consumption standpoint to ensure comprehensive coverage of the possible solutions. Within the research of RBAC (Role Based Access Control) the author suggests that OAuth 2.0 can be used to complement the implementation through the use of OAuth tokens which can be used to identify a subject who has sent a request for access to data and the subsequent access roles related to that subject. The researcher also expands upon the least privilege access principle detailing 2 key areas of importance, the first being access granularity meaning there should be a fine level of control over access, the second area detailed under least privilege access is the ability to grant temporary privileges and revoke them after a session ends or an expiry time passes.

The information provided by (Colombo, Ferrari and Tümer, 2021) around access control approaches and mythologies has been very useful due to the information around different solutions for implementing access control especially as it points out that OAuth 2.0 can be used to implement access control measures which would allow a centralized point of management that would offer a great level of dexterity in deployments due to the fine-grained level of control OAuth supports.

## 2.6 Literature Review reflection

The literature reviews provided valuable insights into different aspects of access control and zero trust principles in the context of IoT systems focused on zero trust and security highlighting many important factors of designing zero trust architecture such as the 7 tenets of zero trust security as outlined by (Rose et al., 2020) and the importance of access control and strong authorisation measures as presented in the works of (Colombo, Ferrari and Tümer, 2021) and (Ameer et al., 2022).   
The purpose of this project is to combine all the theoretical approaches above and create an application that is founded in zero trust and enables IoT communication over large geographical areas and evaluate how zero trust solved the shortcomings of traditional methods of security.

# 3 Methodology

## 3.1 Methodology of Implementation

Zero Trust implementations for IoT (Internet of Things) devices require the consideration of both the security aspect of applications but also the usability of applications, more specifically how authentication, access control, strong user identity and encryption can all be implemented in a meaningful and impactful manner to greater the security posture of an application or organization. This project aims to address each point of zero trust by using opensource and low-cost technologies such as Next.js, Node.js, 0Auth, Tailscale and Python to form the functions necessary to create a secure, authentication-focused application that is centred around zero trust security.

The project’s success will be measured through a group of metrics designed to ensure security implementation’s function as intended such as ensuring WebSocket connections use TLS and API calls are enforced with JWT (JSON Web Tokens), see table 1 below for metrics:

Table 1 - Project metrics

|  |  |  |
| --- | --- | --- |
| **Entity** | **Metric** | **Description** |
| Web Client | Accessibility | Site accessible via VPN-based domain. |
| Web Client | Accessibility | Site supports HTTPS. |
| Web Client | Authentication and Authorization | Requires authentication for functionality access. |
| Web Client | Authentication and Authorization | Authorization is restricted to VPN domain. |
| Web Client | Authentication and Authorization | OAuth login provider operational. |
| Web Client | Authentication and Authorization | User information populates post-login. |
| Web Client | Authentication and Authorization | JWT obtained through OAuth. |
| Web Client | API Security | Unauthorized access to the API endpoint returns as unauthorized. |
| Server | Application Startup | All necessary servers start via startserver.sh. |
| Server | API Server Security | Main API server (monitor.js) correctly validates JWTs. |
| Server | API Server Security | Returns unauthorized to requests not meeting authentication requirements. |
| Server | API Server Security | Only supports HTTPS, rejecting HTTP requests. |
| Server | WebSocket Secure Server | Only allows WSS connections (HTTPS encrypted). |
| Server | WebSocket Secure Server | Restricts connections to VPN network. |
| Server | WebSocket Secure Server | Reacts to commands within a limited command-based structure. |
| Server | WebSocket Secure Server | Correctly dispatches the "updatename:" command. |
| IoT Client | Connectivity | Can connect through VPN domain name. |
| IoT Client | Communication Security | Connects to WSS server over HTTPS connection. |
| IoT Client | Communication Security | Performs TLS handshake with the server. |
| IoT Client | Device Interaction | Retrieves MAC address at runtime. |
| IoT Client | Device Interaction | Responds correctly to "sendinfo", "device\_prop", and "updatename:" commands. |
| Access Control | IoT Client Access | IoT clients via VPN account can only access the server through port 444. |
| Access Control | Web Client Access | Web clients can access the web server and API endpoints through VPN, restricted to ports 3002 and 443. |

To ensure the application can be implemented on remote systems such as the cloud and organisation-owned systems without sacrificing security principles the application will be designed on a VPS (Virtual Private Server) remotely hosted through https://clovux.net in a North American data centre as well as reconfigured inside a locally hosted VM (Virtual Machine) through the use of VMware Workstation Pro 17 as this is the software that Anglia Ruskin University has access to in order to replicate the results. Both systems will use Ubuntu 20.04 however the VPS will use a headless version of Ubuntu, both versions of the operating systems are available at <https://releases.ubuntu.com/focal/> the other reason the project will be rebuilt inside a VM is due to the complex nature of setting up the application with the correct domains, TLS keys and setup of tailscale and 0Auth control panels.

0Auth is to be used to support implementations of authentication and authorisation functions as it is implemented through Okta services with the second largest market share of 27.52% according to (6sense, n.d.), due to Okta having a large market share and low cost of implementation it ensures that the project can be easily translated into processes an organisation can implement efficiently.

Tailscale was chosen over traditional providers such as OpenVPN for 3 main reasons, cost, ease of configuration and open-source alternatives. Tailscale has a higher connection limit before needing to increase plans, the administration console is easier to configure and Tailscale helps support a free open-source alternative named headscale while allowing for self-hosted VPNs to be setup allowing organisations further confidence in software due to the ability to review the code.

Next.js will be used to drive the front end of the application due to the ability to dynamically stream HTML and flexible rendering ability allowing for both client and server-side rendering offloading resource usage onto a client’s machine otherwise known as client-side rendering or CSR.

Node.js will be used to support Secure WebSocket connections to IoT devices using TLS encryption as well as providing the needed API endpoints for the data-driven web application as well as enforcing authentication on API endpoints.

0Auth will be used as an authorisation provider due to its seamless integration with Single-Page applications and other authorisation providers such as Okta used by many large organizations.

TailScale will be used to provide a secure connection to other devices such as from an IoT device to a WebSocket while removing the need to make ports visible to the public by using VPN tunnelling. TailScale will also be used as it has built-in support for Access-Control Lists otherwise known as ACL’s and support for TLS certificates to be created for domains only accessible within the VPN network.

### 3.1.1 Frontend Implementation Approach with Next.js

Next.js offers many advantages over traditional approaches to web-based development allowing for a feature-rich end product due to the native support for CSR, Dynamic HTML Streaming, Route Handlers and Built-in optimization for scripts allowing for fast development times. By utilizing the ability to dynamically update and modify elements on a page, a real-time data-driven application can be created that removes the need to refresh a page to update information displayed such as the connection status of a device or the information that a device is providing through interval-driven API calls.

### 3.1.2 Backend Implementation Approach with Node.js and WebSocket handling

Node.js is what will power the core functions of the application by providing connections to IoT devices through Secure WebSockets and providing data to web clients through the use of JSON Web Token secured endpoint otherwise known as JWT (see section 2.1.2 for further information). The approach for Backend-sever is to have 2 split servers to divide the workload to further support scalability. The first will be for checking a client’s ready status through ICMP pings otherwise known as Internet Control Message Protocol and will solely be for checking if a device is online. The second will have a split functionality which will be to provide WebSocket connections and the needed device information provider endpoints as well as an endpoint for updating client device names remotely.

### 3.1.3 0Auth and JWT’s

0Auth stands for Open Authorization and is the “industry standard for online authorization” as stated by (0Auth, n.d.), it is often already implemented in many organisations as a form of access control.

JWT stands for JSON Web Token and as stated by (auth0.com, n.d.) is the “industry standard RFC 7519 method for representing claims securely between two parties”, JWT’s allow for continuous authentication as the tokens have expiry times and are digitally signed by the issuer (0Auth in the case of this project) and allow for authentication data to be sent within a token.

What does a JWT encode in Base64 look like?

A screenshot of a computer

Description automatically generated

Figure 1- Encoded JWT Token

In figure 2 each part of the token is shown in a different colour to enhance the difference between sections.

The JWT token above in figure 2 contains 3 main sections:

* Header (Red)
* Payload (Pink)
* Verify Signature (Blue)

Each contains important information which can vary depending on what is needed as shown in figure 3, figure 4 and figure 5 below.

A screenshot of a computer

Description automatically generated

Figure 2 - Decoded JWT Header

A screenshot of a computer

Description automatically generated

Figure 3 - Decoded JWT Payload

A screenshot of a computer

Description automatically generated-

Figure 4 - Decoded JWT Signature

JWT’s prevent the need for stateful authentication as the token contains all the information needed to verify a user’s identity and access permissions meaning there is no need for a server to store a session state mitigating the risk of session hijacking-based attacks. Furthermore due to JWT’s being digitally signed a server can verify the authenticity of the token and the data inside the token it receives preventing token forgery attacks. JWT’s also have expiry times which prevent tokens if stolen from working after the expiry. Overall JWT’s create a secure, decentralised authentication method with a fine-grain level of access control without introducing assumed trust following the zero-trust mantra of never trust, always verify.

A diagram of a algorithm

Description automatically generatedA diagram of a software flow

Description automatically generated

Figure 5 - JWT request process

Figure 6 - JWT verification and authentication process

3.1.3.1 What are Digital signatures?

A digital signature is a way to verify the authenticity of data through public key cryptography (asymmetric encryption) it also ensures that the data hasn’t been tampered with or altered after it was signed.

### 3.1.4 Implementation Approach for Tailscale, ACL’s and MagicDNS

Tailscale is a Mesh typology VPN utilising peer-to-peer connections built on top of the WireGaurd protocol which is defined as a “tailnet” as stated by (Tailscale, 2022). Tailscale allows for access to services hosted locally on a machine to be accessed remotely without having to expose ports and services to the public internet limiting potential attack vectors. The service also allows for custom ACL’s to be easily defined with a level of control providing the ability to restrict access to a specific port over a specific protocol with support for all protocols listed by (internet assigned numbers authority, 2024) allowing for further micro-segmentation of resources. Tailscale also supports HTTPS certification provisioning for domains only accessible within the VPN ensuring all traffic is encrypted through the HTTPS protocol as well as through the VPN layer with end-to-end encryption.

3.1.4.1 What is TLS?

TLS stands for Transport Layer Security and is the modern standard that replaced SSL “TLS is a direct successor and replacement for SSL”, TLS is what HTTPS (HyperText Transport Protocol Secure) uses to create secure, encrypted connections with web servers and web-based applications as explained by (Bhat and Mill, 2018).

3.1.4.2 How is TLS intended to be used in the project?

Tailscale is intended to be used as a layer of protection to prevent having to expose web applications and servers to external facing connections while also providing a further layer of encryption on top of the TLS standard.

# 4 Implementation

## 4.1 Network topology

As part of the project, it was important to ensure that the network designed is decentralized and adaptive due to the nature of IoT systems to simulate a realistic scenario where a ZTA-founded approach could be challenging, the network architecture shown in Figure 7 is the same as the architecture used for both the VPS configuration and the VM configuration with the role each device played noted within the diagram such as IoT-Client for simulated IoT devices, Web-Client for machines simulating end-user devices accessing the web application and Server for machines running server and backend applications. It should be noted that due to the microservices architecture of implemented server components server components can be configured to run on individual hardware, VMs or containers to further improve security through micro-segmentation.

A diagram of a computer network

Description automatically generated

Figure 7 - Network typology

### 4.1.1 Network level access control (NAC)

In the current NAC implementation, direct communication between the server and client is intentionally disabled for security reasons. This strategic decision is depicted in figure 7, to enhance security by leveraging Tailscail’s software-defined network (SDN). This approach mitigates the need to expose server ports to the internet. Additionally, by utilising Tailscale’s SDN access control at the free tier accounts used to connect IoT clients to the SDN are restricted to only being able to access the server on the network which is further restricted to only allow access to port 444 which is the port listened to by the Web-Socket server, similar restrictions are also implemented for web-client access restricting access to ports 443 and 3002 which is the port the web application is accessed through and the API server respectively.

### 4.1.2 Tailscale, Tailnet implementation

Using Tailscale for all devices within this project means that all devices form a peer-to-peer network often called a mesh network typology. By doing this it means that all devices are connected directly, forming a private network or what Tailscale calls a “Tailnet”. Inside a Tailnet each device is given an IP address in the CGNAT range (Carrier-Grade Network Address Translation) as shown below in Figure 8.

A screenshot of a computer

Description automatically generated

Figure 8 - Assigned IP addresses in the CGNAT range

A screenshot of a computer program

Description automatically generated

Figure 9 - ACL

Table 2 - ACL breakdown

|  |  |  |  |
| --- | --- | --- | --- |
| tail6349a.ts.net |  |  |  |
| Account | Use case | Destination IP/'s allowed | Destination port/'s allowed |
| [jagsblast@gmail.com](mailto:jagsblast@gmail.com) | IoT devices to web socket server | 100.125.41.124 | 444 |
| [jagsblast1@gmail.com](mailto:jagsblast1@gmail.com) | Web Client access & server | 100.125.41.124 | 443, 3002 |

Implementing connection rules shown in figure 9, helps mitigate potential vulnerabilities from being exposed as an attacker if they were able to gain access to a VPN account and would only be able to interact with ports allowed within the ACL. The second and third features of tailscale use to implement secure communications work together to allow for TLS-encrypted connections to be created. MagicDNS allows for DNS records to be created within the VPN network which can then be used to provision a TLS certificate and key allowing for encryption in transit protocols such as HTTPS to be utilised.

## 4.2 Code Implementations

### 4.2.1 Software level access control

#### 4.2.1.1 0Auth and JWT implementation

To ensure that every request for data is authenticated and that the user requesting the data has the correct access control permissions, access tokens were implemented using JWTs. This process involves two main steps, as illustrated in Figure 5 when a client requests an access token and in Figure 6 when an access token needs to be verified.

Upon loading the web application, users are prompted to log in through the 0Auth login provider as shown in Figure 10.

A screenshot of a login form

Description automatically generated

Figure 10 - 0Auth login provider

Once a user is authenticated through the login provider the web application requests information about the user, such as the user\_id. For example, a user\_id may look like “auth0|65ce207beb8324e1ef01bb0f”. This user\_id is later used to check if a user has the correct permissions to access a resource, known as an audience in the case of 0Auths API.

In the code snippet below shown in Figure 11, the application utilises the @auth0/auth0-react library to communicate with the 0Auth domain specified in the app.js file shown in Figure 12, to check if the logged-in user has the correct permissions ‘read:posts’ as shown in Figure 13 regarding the audience ‘<https://hallmonitor.tail6349a.ts.net/api2/ipMacMap/>’. If the user has the necessary access control permissions, 0Auth’s API will return an access token. This token can then be sent in the header of the request to the application’s API server to pass the authentication and authorisation check.

A screen shot of a computer code

Description automatically generated

Figure 11 - Fetch client access token

A screen shot of a computer screen

Description automatically generated

Figure 12 - App.js Domain & Client ID

A screenshot of a computer

Description automatically generated

Figure 13 - 0Auth Permissions Dashboard

JWTs are used to prevent unauthorised access to the API’s endpoint due to the security features built into JWTs, such as token expiry times and digital signatures which prevent the contents of tokens from being modified as well as meaning they are un-useable after a set time has passed. By implementing 0Auth and JWTs, the application ensures secure access to IoT device data and ensures continuous authentication of users, aligning with the principles of zero trust architecture. JWTs can also be used to provide a higher degree of access control than demonstrated in this project by checking a user’s permissions per scope and requiring individual permissions per API endpoint.

### 4.2.2 Dashboard Development

A screenshot of a computer

Description automatically generatedIn order to provide the data in a way that is easy to understand and extends the utility of the data it was important to make a clear definition of what data correlates to what devices which is why every device has the following information before any device generated specific data shown in Figure 14:

* Name
* ID
* VPN IP
* MAC address
* Device response status
* WebSocket connection status

Figure 14 - Information Card layout

An important use of the data fetched from the API is the way it is used to draw attention to devices with issues such as those not currently connected to the WebSocket server, which when detected will turn the card's background colour from green to red. It was also decided to have 2 steps to verify device readiness as shown in Figure 15, the first step being if the device can be reached through the vpn network through an ICMP (Internet Control Message Protocol) otherwise known as pinging, and the second stage is to verify the device is connected to the web socket server running that handles the data exchange for both the web application and the WebSocket secure server.

A close up of words

Description automatically generated

Figure 15 - Device status indicators

### 4.2.3 IoT device communication

Another important feature implemented is how a user can communicate with individual IoT devices through the dashboard securely. For a user to change the name of a device they must have a valid access token that is validated the same way as when a user requests the of connected devices. Once the API server has the data needed to change the name of an IoT device it is then passed to the Secure Websocket server which invokes the function SendMsgtoClient with the data ipAddress and the message of updatename:${name} where ${name} is replaced with the correct new name as shown below in Figure 16.

A screen shot of a computer program

Description automatically generated

Figure 16 - API to Websocket interaction

The WebSocket server and clients work of a limited command-based structure where pre-designated commands are used to enact important functions, it also means that possible attack vectors are limited due to information passed not being executed and instead compared to strings containing commands. The reason websockets were implemented in this way is that it allows for per-client communication rather than all clients being able to see communications seen with the broadcast method.

Table 3 - Limited command-based structure table:

|  |  |  |  |
| --- | --- | --- | --- |
| Command | Client or Server origin | Notes | Type of Interaction |
| send-info | Server | Sent from the server to the client upon connecting to the server to request the device's Mac address and the name stored in the config,json file.  (this command is issued every 5000 milliseconds in the current program) | Request for data |
| device\_prop | Server | Sent from the server to the client upon connecting to the server to request information the IoT device is collecting as well as the device type such as “air quality”.  (this command is also issued every 5000 milliseconds in the current program) | Request for data |
| updatename: | Server | Sent from the server to the client when a user updates the name of the device in the web application, update name follows the syntax “updatename:[NEW NAME]” and will update the name to whatever is after : in the string. | Update to currently stored data |

A Limited command-based structure helps maintain a level of security by limiting the commands and actions that can be performed thus reducing potential attack vectors while making it easier for future features to be implemented. It also enables per-client communication, preventing all clients from seeing communications intended for one specific client, as might happen with a broadcast method.

#### 4.2.3.1 WebSocket Secure (WSS)

WebSocket communications are carried out over the WebSocket Secure protocol which uses the same encryption standard as HTTPS which is TLS, using WSS ensures that all data exchanged between the WebSocket server and clients is encrypted from end-to-end protecting the integrity and confidentiality of the data in transit. This encryption helps prevent man-in-the-middle attacks such as packet tampering and eavesdropping.

In order to create a WebSocket Secure connection, both a private key and an authorised certificate are required which in the case of this project are obtained from Tailscale as the domain is a private domain only accessible within the VPN network which is one of the reasons Tailscale was selected over other tools such as OpenVPN as it allows for users to provision HTTPS certificates for domains within the VPN network allowing for robust encryption and security for WebSocket communications.

#### 4.2.3.2 Scalability and Reusability

When working to deploy applications and services within an organisation it is important to consider the scalability not just for the application but for future applications, features, and services as well as utilising existing technologies within an organisation where possible to reduce development times. Which is why 0Auth is used in this project to provide authentication as most security-conscious organisations use the platform due to the wide support it provides and ease of integration. Such as in personal experiences where companies such as Experian leverage Okta provided technologies such as 0Auth to provide authenticated access to internal services. When moving to a Zero-Trust approach it is important to utilise existing technologies to gain one of the largest benefits of zero trust which is the reduction in application deployment effort and time. The other benefit to using existing technologies is that it can make the rollout of new infrastructure simpler from an administrative standpoint as administrators already have experience in using the software and rolling out a new service across an entire organisation successfully and effectively. Regarding scalability, both 0auth and tailscale have the required management tools and service availability to ensure that scalability is never an issue as well as being able to restrict access to individual services down to individual endpoints meaning that all user information is accessible in one place rather than having to use different tenants or platforms for a large number of users reducing points of attack as well as reducing operational costs.

## 4.3 Further implementations to Practises and Processes within an organisation for ZTA.

### 4.3.1 Continuous Monitoring and Risk Assessment

In order to maintain a high level of security competency within any organisation it is important to ensure that all current deployments and upcoming deployments are thoroughly assessed for associated risks due to the ever-changing landscape of vulnerabilities within software and hardware. By Continuously monitoring and analysing an organisation’s technology stack the risks presented can be mitigated or resolved using updates or by moving to another provider that offers a more security-oriented approach to the solution required. It is also important to ensure that constant monitoring is utilised to spot abnormalities in the behaviour of devices and the software running on those systems as abnormal spikes in recourse usage or network traffic can hint towards potential malware or attacks such as DDOS attacks, which is when a network attached device is flooded with internet traffic to disrupt services running on a device or even a whole network, as further explained by (Cloudflare, 2019) which highlights IoT devices as a specific attack vector “Exploited machines can include computers and other networked resources such as IoT devices.”.

### 4.3.2 Dynamic Access Control (DAC)

One aim Zero Trust solutions aim to solve is horizontal privilege escalation which can allow an attacker to move from one account to another of the same or similar access level in the hope that a secondary account may have a path to vertical privilege escalation (increase in a level of access). DAC allows for condition-based access authorisation such as needing to be connected to a VPN or meet requirements such as user group, geographic location or security state of a device as mentioned in the Dynamic Access Control Overview for windows server implementations provided by (dknappettmsft et al., 2024).

Further studies also highlighted the use of BBAC (Behaviour -Based Access Control) which evaluates how a user interacts with an application or systems to identify abnormalities in how a legitimate user would interact with a system, and how possible threat actors would interact, the approach taken in research paper produced by (Adler et al., 2014) focuses on monitoring outbound traffic of a device for both “new hosts” as well as “observed behaviour” with a TP (true positive) rate of 92% but also suffering from a FP (False Positive) rate of 20% by using “technique b” concluding that “Neither of these approaches is ideal” and that further work is required to improve the accuracy of the machine learning model which can be improved with larger data sets.

Machine learning plays a crucial role in enhancing BBAC efficiency and effectiveness, particularly with advancements in processing power enabling real-time analysis on large datasets. Continued research and access to larger datasets are essential for refining models to increase accuracy into a state they become more reliable and detailed such as how modern web applications utilise reCAPTCHA to detect non-human website interactions through the use of “an advanced risk analysis engine” as stated by (Google, n.d.) which uses every interaction to increase its dataset. DAC and BBAC are creating the foundation for ML-driven cyber security in a way that could allow for instructions to be detected and segregated to an individual device before any damage is done without human intervention.

### 4.3.3 Micro-Segmentation

Micro-Segmentation attempts to mitigate similar issues as DAC but at a network level, by dividing a network into smaller segments and zones with individual policies such as ACL’s (Access Control Lists) and DAC requirements with the aim to confine any breach into a small zone rather than an entire network. Figure 17 shows a demonstration of a segmented network where each section would have its own subnetwork and from then further configuration on who should have access to what resources would be mapped out and configured in line with a lowest-level privilege model which provides the lowest level access possible to users.

A diagram of a computer network

Description automatically generated

Figure 17 - Micro-Segmented network

Segmentation can be further enhanced, down to a per-user or group-based level of access control within a network through the use of the RADIUS protocol which allows for network-level access control based on the permissions or roles of an account for example if HR and payroll documents are stored in Server section 1 and source code and important business documents are stored in server section 2 a HR employee doesn’t need access to server section 2 so the account wouldn’t hold the necessary permissions to access that section of the network. Implementing network segmentation in this fashion has multiple benefits not just for security but also for setting up new accounts as they only need to be added to a user group to get working.

### 4.3.4 Identity-Centric Security

Identity-centric security is one of the foundational principles of zero-trust architecture, emphasizing the importance of verifying and authenticating a user’s identity before granting access to a resource.

Zero trust solutions attempt to move away from perimeter-defined security based on the methodology that security should remain effective in the event a network is breached and towards focusing on securing networked resources through DAC. Identity-centric approaches aren’t normally used exclusively by themselves but are often paired with access control which is why it is commonly referred to as ICAM, as outlined by (Lam et al., 2020) as “Identity, Credential, and Access Management”. By utilising identity-centric models’ proactive mitigation of internal threats is achieved through implementing least privilege access to only what a user would normally need preventing overlap into other sections of resources without a need for them. Often organisations use tools such as “multifactor authentication” as implemented by (AWS, n.d.) in the “AWS IAM Identity Center”.

Multifactor authentication allows for a secondary device, normally a mobile phone to be used to verify an authentication request as legitimate. Multifactor authentication can be more than just a prompt on a mobile application, it can draw on Biometric factors combined with ownership factors as well as other factors such as geographical factors as well. Using multiple steps or factors of authentication increases the complexity of attacks such as account takeovers as the failure point is no longer weak passwords but a person’s physical identity such as the structure of a person’s face, and unique fingerprints as outlined by (Ometov et al., 2018) in the 3 identified sections of authentication “knowledge factor”, “ownership factor” and “biometric factor”. Many industries where authentication is crucial have already begun developing systems to ensure customers are protected much like how banks have already implemented, voice recognition otherwise called “Person Voice Recognition Methods” in the study by (Tymchenko et al., 2020) which outlines the issues with this approach with the current technology however it showed that by analysing pattens in speech, it is possible to verify a person’s identity due to the difficult nature to falsify a person’s speech and tone patterns.

### 4.3.5 Compliance and Regulatory Requirements

Implementing security measures that meet regulatory and compliance requirements can be complicated when dealing with IoT devices with exponentially increasing complexity when data crosses geographic boundaries regardless of whether the IoT device is collecting the data, processing it or storing it due to ever-evolving regulations and requirements such as EU GDPR (GDPR, 2018) and the UK’s equivalent Data Protection Act otherwise known as DPA (GOV.UK, 2018) due to the varying requirement set out by the legislation. However, by utilising a zero trust approach when creating systems and applications parts of regulations are automatically covered, specifically one of the key seven principles outlined by (Information Commissioner's Office, 2023) “Integrity and confidentiality (security)” due to the nature of zero trust implementations ensuring all sensitive information is encrypted as well as having robust security measures such as access control, network segmentation, and continues risk assessments and monitoring implemented. The same can be said for other compliance standards such as PCI DSS (Payment Card Industry Data Security Standard) which requires many security measures as outlined by (PCI Security Standards Council, 2018) in the “PCI DSS Quick Reference Guide” which has the following requirements, that would be covered under a successful zero trust approach:

* Build and Maintain a Secure Network and Systems
* Regularly Monitor and Test Networks
* Implement Strong Access Control Measures
* Maintain an Information Security Policy

# 5 Results

## 5.1 Summary of Implementation

*For the full implementation of the code refer to https://github.com/jagsblast/Final-Project-2108502/tree/main*

To see whether the implementation of a zero-trust IoT application named hallway monitor was successful a list of key areas must be evaluated first:

Project tests:

Table 4 - Project outcome tests

|  |  |  |
| --- | --- | --- |
| Web client | Server | IoT Client | ACL | Test | Pass | Fail |
| Web client | site can be accessed via vpn based domain (https://hallmonitor.tail6349a.ts.net/) | Pass |
| Web client | site supports HTTPS | Pass |
| Web client | site requires authentication to access functionality | Pass |
| Web client | authorisation can only be granted through vpn domain | Pass |
| Web client | 0auth login provider working correctly | Pass |
| Web client | User information populates post login | Pass |
| Web client | client can obtain JWT through 0auth | Pass |
| Web client | accessing api endpoint without providing a valid JWT returns as unauthorised | Pass |
| Server | application starts all needed servers via startserver.sh | Pass |
| Server | main api server (monitor.js) validates JWT's correctly | Pass |
| Server | main api server (monitor.js) server returns unauthorised to any request that does not meet authentication requirement | Pass |
| Server | main api server (monitor.js) only supports HTTPS and rejects any HTTP request | Pass |
| Server | websocket secure server (monitor.js) only allows for WSS connections (HTTPS encrypted connections) | Pass |
| Server | websocket secure server (monitor.js) only allows for connections from within the vpn network | Pass |
| Server | websocket secure server (monitor.js) only reacts to commands within the Limited command-based structure | Pass |
| Server | websocket secure server (monitor.js) correctly dispatches the command "updatename:" | Pass |
| IoT Client | IoT client script can connect through vpn domain name | Pass |
| IoT Client | connects to WSS server over https connection | Pass |
| IoT Client | IoT client performs TLS handshake with server | Pass |
| IoT Client | IoT client script gets mac address at runtime | Pass |
| IoT Client | IoT client script responds to "sendinfo" correctly | Pass |
| IoT Client | IoT client script responds to "device\_prop" correctly | Pass |
| IoT Client | IoT client script responds to "updatename:" correctly | Pass |
| ACL | IoT Clients connected through IoT client vpn account can only access the server and only through port 444 | Pass |
| ACL | Web Clients can only access the web server and api endpoints through the vpn and only through port 3002 and 443 | Pass |

### 5.1.1 IoT Client

#### 4.1.1.1 Client Sever communication integrity.

The Domain of server hallmonitor1.tail1e215.ts.net translates to the IP address of 100.125.41.124 through Tailscales MagicDNS feature.



Figure 18 - Client initiates TLS handshake



Figure 19 - Server response to TLS handshake (Server Hello, Key exchange, Cipher Spec)

A screenshot of a computer

Description automatically generated

Figure 20 - Detailed Server response to TLS handshake (Server Hello, Key exchange, Cipher Spec)

To ensure that data sent between the client and server is protected the TLSv1.3 protocol is used to encrypt traffic at the 4th layer (the Transport layer) providing encryption authentication and integrity protection for data transferred between the client and server within the OSI model outlined in the overview provided by (Kumar, Dalal and Dixit, 2014). Tailscale provides further insurance of data integrity by providing a secure channel for secure communication at the layer 3 level (the Network level) by providing a secure network infrastructure for connections and data transmission that TLS can operate on top of.

Figure 18 shows the first step of a TLS handshake from the client (100.94.44.90) to the server (100.117.186.96) which is followed by the response in figure 19 from the server which when examined contains 2 components of a TLS handshake, Hello, Change Cipher Spec which can be seen in a detailed view as shown in Figure 20 which shows the server's acknowledgement (ACK) packet along with a cipher spec message which notifies the client that all further communications will be protected under the negotiated Cipher Spec in this case TLS\_AES\_256\_GCM\_SHA384 with the short name of 1302 as defined in “Table 1. TLS ciphers” provided by (IBM, 2023).

#### 5.1.1.2 IoT client script functionalities

When the script first starts it attempts to get the MAC (Media Access Control) address of the network interface of the device through the python library UUID (Python Software Foundation, 2024) that provides the function “uuid.getnode()” in accordance with RFC 4122 which produces a mac address containing 12 bits in hexadecimal notation separated by : every 2 bits that looks like 00:00:00:00:00:00.

In the implementation of this client the data that is passed to the server is static data stored in a JSON file due to not having a source for live data however this can be easily substituted by modifying the client script,

Config.json:

{

  "name": "air quality monitor #1",

  "Device\_prop": [

    {

      "type": "air quality",

      "info\_given": [

        {

          "bar": [

            "Oxigen: 55% info",

            "CO2: 30% danger",

            "Other: 15% warning"

          ]

        }

      ]

    }

  ]

}

The client script is then able to read the file and process the contents of the file into the correct variables that are later sent to the server over the WebSocket secure connection when requested via the limited command structure references above in section 4.4.1.

Data population code:

 device\_properties = config\_data.get("Device\_prop", [])

    device\_type = device\_properties[0].get("type", "Type not specified") if device\_properties else "Type not specified"

    Device\_info\_given = device\_properties[0].get("info\_given", "Type not specified") if device\_properties else "Type not specified"

After a client has established a secure connection with the server is sends the device MAC address as well as the name of the device set in the config,json file as shown in the code snipped below where the script utilises async functions to wait until the connection is established to send the information:

The initial transmission of device information:

async with websockets.connect(uri) as websocket:

                print("Connected to the server")

                # Send MAC and name information to the server

                await websocket.send(f"mymac: {mac}, myname: {name}")

After establishing the connection and the server has assigned the connection to the provided MAC address and name, it then sends a command to the client to request further information which is handled by the following code resulting in a response as shown in Figure 21 and Figure 22 respective of what is requested.



Figure 21 - Response to sendinfo command



Figure 22 - Response to device\_prop command

Command handling code:

 try:

                        # Receive messages from the server

                        message = await websocket.recv()

                        print(f"Received from server: {message}")

                        # Check if the message matches a command

                        if message == 'sendinfo':

                            await websocket.send(f"info: mymac: {mac}, myname: {name}")

                        elif message == 'device\_prop':

                            await websocket.send(f"device\_type: {device\_type} Device\_info\_given: {Device\_info\_given}")

                            config\_data = get\_config\_data()

                            device\_properties = config\_data.get("Device\_prop", [])

                            device\_type = device\_properties[0].get("type", "Type not specified") if device\_properties else "Type not specified"

                            Device\_info\_given = device\_properties[0].get("info\_given", "Type not specified") if device\_properties else "Type not specified"

                            print(device\_type)

                            print(Device\_info\_given)

                        elif message.startswith("updatename:"):

                            split\_msg = message.split(":")

                            updated\_name = split\_msg[1]

                            write\_name\_to\_config(updated\_name, device\_properties)

                            config\_data = get\_config\_data()

                            name = config\_data.get("name", "YourNameHere")

Furthermore, another important implementation within the client script is the ability to automatically reconnect to the server if the connection is closed, this is why an auto-reconnection function was added to ensure that the client attempts to reconnect every 5 seconds if the connection is closed which is why if connection error occurs the code below catches the error and attempts to create a new connection to the server as shown in Figure 23.

Auto-reconnection code:

        except (websockets.WebSocketException, ConnectionRefusedError):

            print("Unable to connect. Retrying in 5 seconds...")

            await asyncio.sleep(5)

if \_\_name\_\_ == "\_\_main\_\_":

    asyncio.get\_event\_loop().run\_until\_complete(connect\_to\_server())



Figure 23 - Auto-Reconnection function output

### 5.1.2 Server-Side Functionality overview and results

The server implementation is the core of this project providing both a central secure point for IoT devices to connect to, as well as driving the web application through 0auth integration and API availability. In the Backend of the web application, there is a total of 3 nodejs servers one which provides a web client built with react and nextjs and the other two provide high availability API endpoint servers one of which also handles WebSocket connections.

Table 5 - Server functionality providers

|  |  |  |  |
| --- | --- | --- | --- |
| Server name | What the server provides | File name | Requires Authentication? |
| Monitor | Websocket Secure connections, API endpoints for:   * /api2/ipMacMap * /api2 * /api2/updatename/:ipAddress | Monitor.js | Yes for all endpoints |
| Ping Server | API endpoints for:   * /api/ping /:ipAddress | pingServer.js | No, due to lack of need and need for high availability |
| App | Hosts web application accessible in a browser once connected to VPN. | App.js | Yes, once web application has loaded |

#### 5.1.2.1 Monitor results

The implementation of Monitor resulted in successful API authentication and authorization enforcement for all API routes under /api2 via the use of JWT’s along with the restrictions on connections implemented through the use of ACL’s within the VPN. Along with the provided security of using a VPN that encrypts traffic, the HTTPS protocol was also successfully implemented for both the API endpoints and WebSocket connections as shown in the packet capture in figure 20 which shows a detailed view of a TLS handshake.

Several metrics were outlined that must be passed for the implementation to be marked as a success as outlined in Table 2 – Project outcome tests, each of which passed without issues. The tests included both functionality requirements as well as security requirements such as allowing only connections from within the VPN network, which was surpassed by blocking requests deemed unauthorised as well as further implementing restrictions on allowed connections within the VPN network through ACL’s.

# 6 Discussion

## 6.1 Evaluation of Zero Trust implementation

In this project, a zero-trust approach was used to develop a web-based application to support the secure implementation of IoT devices by using zero-trust best practices. The architecture implemented covers access control, micro-segmentation, continuous authentication and strong identity verification by using multiple steps to ensure the risk of user impersonation and credential theft are mitigated along with other attacks such as man-in-the-middle attacks.

Data in transit is protected at multiple layers within the OSI model. Tailscale utilises the WireGaurd protocol, operating at the 3rd layer within the OSI model to create a secure virtual overlay network (otherwise known as VON) and protects data using cryptography methods such as “Noise protocol framework, Curve25519, ChaCha20, Poly1305, BLAKE2, SipHash24, HKDF” as stated by (Donenfeld, 2015). Furthermore, when ether an IoT or Web client initialise a connection the server and client initiate a TLS handshake to provide TLS encryption which operates within layer 4 of the OSI model.

Zero-trust practices such as least privilege access, micro-segmentation and continuous authentication were implemented by using services providers 0Auth and Tailscale through the use of role-based access and access control lists to allow fine-grain control over access to resources. The current implementation isn’t best optimised for micro-segmentation or high availability requirements due to limited resource capacity however due to the design of the application individual services can be relocated to separate resources and corrections made to ACL’s with ease allowing greater flexibility for implementation. Due to the module-centric design of the application micro-segmentation could be improved by relocating services to individual virtual machines allowing for self-contained services and preventing further disruptions in the event an individual service is breached.

The security posture achieved within the project would not be possible with traditional security architectures due to the complexity of the created network architecture as approaches such as perimeter-based security as the project makes allowances for remote IoT devices which empowers organisations to collect greater amounts of data from different geographical locations securely.

## 6.2 Comparison with Traditional Security Models

Within cyber security, there are many security models each with advantages and disadvantages that help protect against specific threat models.

Table 6 - Security Model pros and cons

|  |  |  |
| --- | --- | --- |
| Security Model | Pros | Cons |
| Perimeter-Based Security Model | Simple model typology | Lack of internal protections |
|  | A clear boundary between internal and external network | Not viable with remote working implementations |
|  | Provides a basic level of protection against external threats | Often fails due to a lack of internal controls |
|  |  |  |
| Castle-and-Moat Model | Centralised management of network security | Limited access control restrictions |
|  | Historical success of preventing external attacks | Lack of Insider threat protections |
|  | Easily built on to improve security | Single point of failure of protection |
|  |  |  |
| Zero-Trust Security Model | High degree of control for access control | Complex architecture makes implementation lengthy and difficult |
|  | Continuous monitoring of resources | Potential for misconfiguration due to the complexity of implementations |
|  | Increase compliance with regulations and ease of supporting new requirements | Due to high-level monitoring and continuous authentication management and validation resource usage tends to be high |

In table 5 the zero-trust security model and 2 traditional modelled were reviewed to compare how zero-trust security models stand up to historical methods of implementing security practices within a network and how they fall short of what is required in the modern day.

For example, both the Perimeter-Based Security models and Castle-and-Moat Model fail to meet the requirements needed in the current times of remote working and cloud-based computing due to focusing on only maintaining a network's perimeter. However, zero-trust was built from the ground up with cloud computing and remote working in mind which is demonstrated through the emphasis on continuous authentication along with other methodologies such as micro-segmentation and least privilege access through RBAC and DAC.

# 7 Conclusion

## 7.1 Summary and evaluation of research and implementation.

In conclusion, this project has evaluated historical approaches to security as well as currently developing methods such as zero trust to provide a holistic overview of the shortcomings of past methodologies and how zero trust approaches work to mitigate past issues and support the development of cutting-edge technologies allowing organisations to become agile in their approach to security for networks and applications in response to evolving cyber threats.

The primary aim was to evaluate how a zero-trust approach can benefit security when implementing devices outside an organisation's normal network over traditional security measures that focus on perimeter-based security. After reviewing publications such as NIST Special Publication 800-207 and technologies currently available such as Tailscale, it was found that by moving away from perimeter-based solutions and towards an approach that focuses on concepts such as access control, micro-segmentation, continuous authentication and strong identity principles such as zero-trust, implementing devices at remote locations or outside of an organisation's normal network was achievable and provides a stronger security posture for systems in the event of an attack.

The Secondary aim was to design and implement a secure platform founded in zero trust to enable secure access to IoT device data. Risk mitigation techniques such as strategic choices to only enable connectivity through VPNs to prevent the need to expose services to the internet and ensure communications between clients and servers utilised encryption standards such as TLS to prevent attacks such as packet sniffing were implemented along with authentication and claim management techniques to ensure access to data was authorised by authentication providers such as 0Auth enabling API endpoints to be secured requiring authorisation for every request.

Overall, the project highlighted the significance and complexity of implementing zero-trust practices, particularly with existing infrastructure. It has also underlined the importance of ensuring security policies and procedures are agile and adaptive due to the evolving landscape of cyber threats. The project has also shown how different approaches can achieve the same goal as shown by the 2 scenarios of implementation undertaken within the project implementing the project on both owned hardware and rented hardware often referred to as the cloud.

## 7.2 Future work and Difficulties

During the project, many issues were presented in the design, development and testing phases requiring innovative approaches to resolve. One example was a visual issue with the graphical side of the application resulting in device cards flickering when updating which was resolved by caching data and implementing loading states waiting for data to be propagated. Another challenge presented during the development was the lack of experience with middleware resulting in excessive time spent implementing functions to retrieve JWT’s from 0Auth API endpoints.

In future work, the researcher would want to implement functionality that verifies device integrity of IoT devices to detect breaches and trigger response mechanisms to resolve such issues. Another feature that the researchers would like to develop is an addition to the server implementation that monitors connections for suspicious or abnormal behaviour and potentially integrate machine learning into anomaly detection for both network traffic and user interaction to enhance DAC features.

# 8 References

0Auth (n.d.). *What is OAuth 2.0 and what does it do for you?* [online] Auth0. Available at: https://auth0.com/intro-to-iam/what-is-oauth-2 [Accessed 15 Mar. 2024].

6sense (n.d.). *Okta - Market Share, Competitor Insights in Identity And Access Management*. [online] 6sense. Available at: https://6sense.com/tech/identity-access-management/okta-market-share [Accessed 5 Apr. 2024].

Adler, A., Mayhew, M.J., Cleveland, J., Atighetchi, M. and Greenstadt, R. (2014). *Using Machine Learning for Behavior-Based Access Control: Scalable Anomaly Detection on TCP Connections and HTTP Requests*. [online] ieeexplore.ieee.org. Available at: https://ieeexplore.ieee.org/abstract/document/6735899 [Accessed 18 Mar. 2024].

Ameer, S., Gupta, M., Bhatt, S. and Sandhu, R. (2022). BlueSky. *Proceedings of the 27th ACM on Symposium on Access Control Models and Technologies*. doi:https://doi.org/10.1145/3532105.3535020.

auth0.com (n.d.). *JWT.IO*. [online] Jwt.io. Available at: https://jwt.io/ [Accessed 19 Mar. 2024].

AWS (n.d.). *IAM - Multi-factor Authentication*. [online] Amazon Web Services, Inc. Available at: https://aws.amazon.com/iam/features/mfa/ [Accessed 20 Mar. 2024].

Bhat, A. and Mill, E. (2018). *https/pages/technical.md at master · GSA/https*. [online] GitHub. Available at: https://github.com/GSA/https/blob/master/pages/technical.md [Accessed 17 Mar. 2024].

Cloudflare (2019). *What is a DDoS attack?* [online] Cloudflare. Available at: https://www.cloudflare.com/en-gb/learning/ddos/what-is-a-ddos-attack/ [Accessed 17 Mar. 2024].

Colombo, P., Ferrari, E. and Tümer, E.D. (2021). *Access Control Enforcement in IoT: state of the art and open challenges in the Zero Trust era*. [online] IEEE Xplore. doi:https://doi.org/10.1109/TPSISA52974.2021.00018.

dknappettmsft, v-kents, eross-msft, sudeepku, yishengjin1413 and billmath (2024). *Dynamic Access Control Overview*. [online] learn.microsoft.com. Available at: https://learn.microsoft.com/en-us/windows-server/identity/solution-guides/dynamic-access-control-overview [Accessed 18 Mar. 2024].

Donenfeld, J.A. (2015). *WireGuard: fast, modern, secure VPN tunnel*. [online] Wireguard.com. Available at: https://www.wireguard.com/ [Accessed 2 Apr. 2024].

FORRESTER (2021). *The Total Economic ImpactTM Of Zero Trust Solutions From Microsoft Cost Savings And Business Benefits Enabled By Microsoft’s Zero Trust Solutions*. [online] Available at: https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RWRIEi [Accessed 1 Mar. 2024].

GDPR (2018). *General data protection regulation (GDPR)*. [online] General Data Protection Regulation (GDPR). Available at: https://gdpr-info.eu/ [Accessed 20 Mar. 2024].

Google (n.d.). *reCAPTCHA*. [online] reCAPTCHA. Available at: https://www.google.com/recaptcha/about/ [Accessed 18 Mar. 2024].

GOV.UK (2018). *Data Protection Act 2018*. [online] Legislation.gov.uk. Available at: https://www.legislation.gov.uk/ukpga/2018/12/contents/enacted [Accessed 20 Mar. 2024].

IBM (2023). *SSL cipher specifications*. [online] www.ibm.com. Available at: https://www.ibm.com/docs/en/ibm-http-server/9.0.5?topic=options-ssl-cipher-specifications [Accessed 21 Mar. 2024].

Information Commissioner's Office (2023). *A guide to the data protection principles*. [online] ico. Available at: https://ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/data-protection-principles/a-guide-to-the-data-protection-principles/ [Accessed 20 Mar. 2024].

internet assigned numbers authority (2024). *Protocol Numbers*. [online] www.iana.org. Available at: https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml [Accessed 1 Apr. 2024].

Kumar, S., Dalal, S. and Dixit, V. (2014). *THE OSI MODEL: OVERVIEW ON THE SEVEN LAYERS OF COMPUTER NETWORKS* . [online] THE\_OSI\_MODEL\_OVERVIEW\_ON\_THE\_SEVEN\_LAYERS-607-libre.pdf. Available at: https://www.researchpublish.com/upload/book/THE%20OSI%20MODEL%20OVERVIEW%20ON%20THE%20SEVEN%20LAYERS-607.pdf [Accessed 21 Mar. 2024].

Lam, T., J Clancy, T., T. Ranks, P., W. Wilmer III, J. (Jack), Department of Defense and Office of the Chief Information Officer (DoD CIO) (2020). *DoD EnterpriseIdentity, Credential, and Access Management (ICAM) Reference Design*. [online] DoD\_Enterprise\_ICAM\_Reference\_Design.pdf. Available at: https://dodcio.defense.gov/Portals/0/Documents/Cyber/DoD\_Enterprise\_ICAM\_Reference\_Design.pdf [Accessed 28 Mar. 2024].

Ometov, A., Bezzateev, S., Mäkitalo, N., Andreev, S., Mikkonen, T. and Koucheryavy, Y. (2018). Multi-Factor Authentication: A Survey. *Cryptography*, [online] 2(1), p.1. doi:https://doi.org/10.3390/cryptography2010001.

PCI Security Standards Council (2018). *PCI DSS Quick Reference Guide Understanding the Payment Card Industry Data Security Standard version 3.2.1 For merchants and other entities involved in payment card processing*. [online] Available at: https://listings.pcisecuritystandards.org/documents/PCI\_DSS-QRG-v3\_2\_1.pdf [Accessed 20 Mar. 2024].

Petrosyan, A. (2023). *Annual number of IoT attacks global 2022*. [online] Statista. Available at: https://www.statista.com/statistics/1377569/worldwide-annual-internet-of-things-attacks/ [Accessed 1 Mar. 2024].

Python Software Foundation (2024). *uuid — UUID objects according to RFC 4122 — Python 3.9.5 documentation*. [online] docs.python.org. Available at: https://docs.python.org/3/library/uuid.html [Accessed 21 Mar. 2024].

Rose, S., Borchert, O., Mitchell, S. and Connelly, S. (2020). Zero Trust Architecture. *Zero Trust Architecture*, [online] 800-207(800-207). doi:https://doi.org/10.6028/nist.sp.800-207.

Tailscale (2022). *What is Tailscale?* [online] Tailscale. Available at: https://tailscale.com/kb/1151/what-is-tailscale [Accessed 19 Mar. 2024].

Tymchenko, O., Havrysh, B., Tymchenko, O.O., Khamula, O., Kovalskyi, B. and Havrysh, K. (2020). *Person Voice Recognition Methods*. [online] ieeexplore.ieee.org. Available at: https://ieeexplore.ieee.org/abstract/document/9204023 [Accessed 20 Mar. 2024].

Vailshery, L. (2023). *IoT Connected Devices Worldwide 2019-2030*. [online] Statista. Available at: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/ [Accessed 5 Mar. 2024].

# Appendix I – Application setup

It is advised that the preconfigured server and IoT client virtual machines are downloaded rather than setting up the application for compatibility reasons also due to cloud storage restrictions the server VM will also act as the web-client VM.

## Preconfigured Virtual Machine setup:

Server VM download link: <https://drive.google.com/file/d/1ipLG_QoM7orO6ZPq3s_ZyjU-aG8mfjJd/view?usp=drive_link>

IoT Client VM download link: <https://drive.google.com/file/d/1Ope0fAV6SR0-naQVBWfNg_YpUdgtyGNs/view?usp=drive_link>

Credentials:

|  |  |  |
| --- | --- | --- |
| Login use | Username: | Password (login and root access): |
| Ubuntu Server | ubu | root |
| Kali IoT-Client | kali | kali |
| Web-page login | jagsblast@gmail.com | Pass2108502 |

Configuration of server:

Open terminal with shortcut CTRL + SHIFT + T and run each of the following commands one after another in the order presented below:

Sudo tailscale up

cd /var/web-apps/hallway-monitor-mainv2/startup

sudo ./startserver.sh (you may be prompted to enter a root password please see credentials table)

Open IoT-Client VM and open a terminal, and run each of the following commands one after another in the order presented below:

cd /home/kali/Desktop/iot/

sudo tailscale up (you may be prompted to enter a root password please see credentials table)

sudo python3 monitor.py

Return to Ubuntu Server VM and open the browser in the server VM and visit the URL below:

Web application URL : <https://hallmonitor1.tail1e215.ts.net>

Login with the login provided in the credentials tables above under login use “Web-page login”.

## Self-setup:

Clone Server and IoT repository from GitHub: <https://github.com/jagsblast/Final-Project-2108502/tree/main>

Once the repository is downloaded, follow the instructions at <https://github.com/jagsblast/Final-Project-2108502/blob/main/README.md>

Or see below for instructions:

Final-Project-2108502 setup:

(step 1) install tailscale from <https://tailscale.com/download>

(step 2) install nvm using curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash from the offical NVM github repo <https://github.com/nvm-sh/nvm>

(step 3) generate a TLS certificate and key using the command tailscale cert [tailscale domain]

(step 3.5) open the file monitor.js and locate var options = { and replace the current paths for both the .key and .cert files to the directory of the files created in step 3

(step 4) install nginx using sudo apt install nginx and use the following config for the default config:

server {

listen 443 ssl;

server\_name hallmonitor1.tail1e215.ts.net;

ssl\_certificate [replace with the full path of tailscale cert created in step 3];

ssl\_certificate\_key [replace with the full path of tailscale key created in step 3];

location / {

proxy\_pass http://localhost:3000; # Change the port if your app runs on a different port

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection 'upgrade';

proxy\_set\_header Host $host;

proxy\_cache\_bypass $http\_upgrade;

# try\_files $uri /index.html; # This line is commented out since it's already handled by the React app

}

# Proxy requests to port 3001 for all API endpoints

location /api {

proxy\_pass http://localhost:3001;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection 'upgrade';

proxy\_set\_header Host $host;

proxy\_cache\_bypass $http\_upgrade;

}

# Proxy requests to port 8000 for another API

location /api2 {

proxy\_pass http://localhost:444;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection 'upgrade';

proxy\_set\_header Host $host;

proxy\_cache\_bypass $http\_upgrade;

}

}

(step 5) restart nginx using "sudo systemctl restart nginx"

(step 6) open a new terminal otherwise NVM will NOT work, install node v18.16.0 using the command nvm install 18.16.0 followed by nvm use 18.16.0

(step 7) open the main directory containing the file package.json and run the command sudo npm install

(step 8) replace both the domain and clientID variables in the index.js file with the correct values for your 0Auth application

(step 9) create .env file at the root of application **note not all values are implomented due to compatibility issues**

# monitor.js variables:

audience = https://[replace with 0auth application domain]/api/v2/ #also used for app.js

issuerBaseURL = https://[replace with 0auth application domain]/

key = [replace with full path of tailscale cert key]

cert = [replace with full path of tailscale cert cert]

corsOrig = [replace with full tailscale url]

# end of monitor.js variables

# start of ipMac.js variables

monitorGetmac=https://[replace with tailscale url]:444/api2/ipMacMap

monitorUpdateName="https://[replace with tailscale url]:444/api2/updatename/"

# end of ipMac.js variables

# start of app.js variables

domain = [replace with 0auth application domain]

clientId = [replace with 0auth application ClientID]

# end of app.js variables

to run servers use startservers.sh in ./startup **it must be ran under sudo due to using port 444** command sudo ./startservers.sh

# Appendix II – Main code and code referred in main text

## Monitor.js code:

const fs = require('fs');

const http = require('http');

const WebSocket = require('ws');

const https = require('https');

const express = require('express');

const app = express();

const { auth } = require('express-oauth2-jwt-bearer');

const cors = require('cors');

const { expressjwt: jwt } = require('express-jwt');

const jwksRsa = require('jwks-rsa');

const bodyParser = require('body-parser');

const { requiredScopes } = require('express-oauth2-jwt-bearer');

require('dotenv').config({ path: '/var/web-apps/hallway-monitor-mainv2/.env' });

const PORT\_WS = 444;

const PORT\_API = 3002;

const checkJwt = auth({

audience: process.env.audience,

issuerBaseURL: process.env.issuerBaseURL,

});

app.use(function (err, req, res, next) {

if (err.name === 'UnauthorizedError') {

res.status(401).send('invalid token...');

}

});

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({

extended: true

}));

var options = {

key: fs.readFileSync(process.env.key),

cert: fs.readFileSync(process.env.cert)

};

const corsOptions = {

origin: process.env.corsOrig,

allowedHeaders: 'Content-Type,Authorization',

optionsSuccessStatus: 204,

};

app.use(cors(corsOptions));

app.use(express.json());

app.use((err, req, res, next) => {

if (err.name === 'InvalidTokenError') {

res.status(401).json({ error: 'Unauthorized' });

} else {

next(err);

}

});

const ipMacMap = {};

const dev\_info = {};

const clients = new Set();

app.get('/api2/ipMacMap', checkJwt, (req, res) => {

res.status(200).json(ipMacMap)

});

app.get('/api2', checkJwt, (req, res) => {

res.status(200).send("authorise");

});

app.post(`/api2/updatename/:ipAddress`, checkJwt, cors(corsOptions), (req, res) => {

const ipAddress = req.params.ipAddress;

const name = req.body.name;

console.log(req.body.name)

console.log('IP Address:', ipAddress);

console.log('Name:', name);

const msg = `updatename:${name}`;

sendMsgtoClient(ipAddress, msg);

res.status(200).send('Name updated successfully');

});

const serverAPI = https.createServer(options, app);

serverAPI.listen(PORT\_API);

const wss = new WebSocket.Server({ noServer: true });

const allowedVpnIps = ['100.64.0.0/10', '127.0.0.1/23'];

const isAllowedIp = (ip) => {

return allowedVpnIps.some((allowedIp) => {

const [allowedNetwork, subnet] = allowedIp.split('/');

const ipSections = ip.split('.');

const allowedSections = allowedNetwork.split('.');

return ipSections.slice(0, subnet / 8).join('.') === allowedSections.slice(0, subnet / 8).join('.');

});

};

const pingInterval = setInterval(() => {

wss.clients.forEach((client) => {

if (client.isAlive === false) {

console.log(`Client ${client.\_socket.remoteAddress.replace('::ffff:', '')} is not responding. Terminating.`);

return client.terminate();

}

client.isAlive = false;

client.ping();

});

}, 20000);

const serverWS = https.createServer(options);

serverWS.listen(PORT\_WS);

serverWS.on('upgrade', (request, socket, head) => {

wss.handleUpgrade(request, socket, head, (ws) => {

wss.emit('connection', ws, request);

});

});

wss.on('connection', (ws, req) => {

console.log('Client connected');

const ipAddr = req.connection.remoteAddress.replace('::ffff:', '');

if (!isAllowedIp(ipAddr)) {

console.log(`Unauthorized connection from ${ipAddr}. Closing connection.`);

ws.terminate();

return;

}

clients.add(ws);

ws.send('send-info');

ipMacMap[ipAddr] = {

connected: 'true',

};

broadcast(`connected: ${ipAddr}`);

const sendInfoInterval = setInterval(() => {

if (ws.readyState === WebSocket.OPEN) {

ws.send('sendinfo');

ws.send('device\_prop')

}

}, 5000);

ws.on('message', async (message) => {

msg1 = `${message}`;

console.log(msg1)

if (msg1.slice(0, 5) === "info:") {

match = msg1.match(/mymac: ([^,]+), myname: ([^]+)/);

ipMacMap[ipAddr] = {

mac: match[1],

name: match[2],

connected: true,

};

} else if (msg1.slice(0, 12) === "device\_type:") {

const pattern = /device\_type:\s\*([^,]+)\s\*Device\_info\_given:\s\*\[([^\]]+)\]/i;

const match = msg1.match(pattern);

if (match !== null) {

const deviceType = match[1].trim();

const deviceInfoGiven = match[2].split(',').map(item => item.trim());

ipMacMap[ipAddr]["dev"] = {

ipAddr,

device\_type: deviceType,

Device\_info\_given: deviceInfoGiven

};

} else {

console.log("Pattern did not match.");

}

} else if (msg1 === 'close-connection') {

console.log(`Client ${ipAddr} disconnected ungracefully.`);

ws.terminate();

}

});

ws.on('pong', () => {

ws.isAlive = true;

});

ws.on('close', () => {

console.log('Client disconnected');

const isStillConnected = clients.has(ws);

if (isStillConnected) {

ipMacMap[ipAddr].connected = 'false';

broadcast(`disconnected: ${ipAddr}`);

}

clients.delete(ws);

clearInterval(sendInfoInterval);

});

});

const broadcast = (message) => {

wss.clients.forEach((client) => {

if (client.readyState === WebSocket.OPEN) {

client.send(message);

}

});

};

function sendMsgtoClient(ip, message) {

wss.clients.forEach((client) => {

if (client.\_socket.remoteAddress.replace('::ffff:', '') === ip) {

client.send(message);

console.log(`Sent message to ${ip}: ${message}`);

}

});

}

## pingServer.js code:

const express = require('express');

const ping = require('ping');

const cors = require('cors'); // Import the cors middleware

const app = express();

const port = 3001;

// Enable CORS for all routes

app.use(cors());

app.use((req, res, next) => {

res.setHeader('Content-Security-Policy', "default-src 'none'; font-src \*;");

next();

});

app.get('/api/ping/:ipAddress', async (req, res) => {

const ipAddress = req.params.ipAddress;

try {

const result = await ping.promise.probe(ipAddress);

res.json({ alive: result.alive });

} catch (error) {

console.error('Error pinging the host:', error);

res.status(500).json({ error: 'Internal Server Error' });

}

});

app.listen(port, () => {

console.log(`Server is running on http://localhost:${port}`);

});

## App.js code:

// App.js or Routes.js

import React from 'react';

import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

import { Auth0Provider } from '@auth0/auth0-react';

import Login from './Login';

import { getAllMac, getMac, updateConnectionStatus, updateName } from './ipMac'; // Import functions

require('dotenv').config({ path: '../.env' });

const domain = process.env.domain;

const clientId = process.env.clientId;

const App = () => {

return (

<Router>

<Auth0Provider

domain={domain}

clientId={clientId}

redirect\_uri={window.location.origin}

audience= {process.env.audience}

scope = { "read:current\_user update:current\_user\_metadata" }

redirectUri={window.location.origin}

>

<Switch>

<Route exact path="/login" component={Login} />

{/\* Add other routes for application \*/}

</Switch>

</Auth0Provider>

</Router>

);

};

export default App;

## page.tsx code:

"use client"

import React, { useState, useEffect } from 'react';

import { useAuth0, Auth0Provider } from '@auth0/auth0-react';

import Profile from './scripts/profile'; // Import the Profile component

import { AddCard, addContent } from './scripts/add-card';

import { getAllMac } from './scripts/ipMac';

import debounce from 'lodash/debounce';

import Posts from './components/posts';

const Home = () => {

const { isLoading, isAuthenticated, user, getAccessTokenSilently } = useAuth0();

const [additionalContent, setAdditionalContent] = useState([]);

const [macData, setMacData] = useState({});

const [isReady, setIsReady] = useState(false);

const [token, setToken] = useState('');

useEffect(() => {

const fetchData = async () => {

const initialMacData = await getAllMac(token); // Pass token here

console.log(token)

setMacData(initialMacData);

const newContent = await addContent([], handleRevokeClick, token);

setAdditionalContent(newContent);

setIsReady(true);

};

const fetchToken = async () => {

try {

const accessToken = await getAccessTokenSilently();

setToken(`Bearer ${accessToken}`);

} catch (error) {

console.error('Error fetching access token:', error);

}

};

if (!isLoading) {

fetchData();

fetchToken();

}

}, [isLoading]);

useEffect(() => {

if (isReady) {

const intervalId = setInterval(async () => {

console.log(token)

const updatedMacData = await getAllMac(token); // Pass token here

setMacData(updatedMacData);

const newContent = await addContent([], handleRevokeClick, token);

setAdditionalContent(newContent);

}, 10000);

return () => clearInterval(intervalId);

}

}, [isReady, token]);

const handleRevokeClick = (cardId) => {

setAdditionalContent((prevContent) => prevContent.filter((card) => card.id !== cardId));

};

const renderContent = () => {

if (!isAuthenticated) {

return <LoginPage />;

}

if (!isReady) {

return <div>Loading...</div>;

}

return (

<main className="flex min-h-screen flex-col justify-between p-24">

<div>

<div>

<Profile /> {/\* Render the Profile component here \*/}

<LogoutButton />

</div>

<div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 gap-5">

{additionalContent.map((content) => (

<React.Fragment key={content.id}>{content.jsx()}</React.Fragment>

))}

</div>

</div>

</main>

);

};

return renderContent();

};

const LoginPage = () => {

const { loginWithRedirect } = useAuth0();

return (

<button className="focus:outline-none text-white bg-blue-700 hover:bg-red-800 w-20 rounded-full m-5" onClick={() => loginWithRedirect()}>Log In</button>

);

};

const LogoutButton = () => {

const { logout } = useAuth0();

return (

<button className="focus:outline-none text-white bg-red-700 hover:bg-red-800 w-20 rounded-full m-5" onClick={() => logout({ returnTo: window.location.origin })}>

Log Out

</button>

);

};

const AuthenticatedApp = () => {

// Check if window is defined before accessing window.location.origin

const redirectUri = typeof window !== 'undefined' ? window.location.origin : null;

return (

<Auth0Provider

domain="dev-bp1s3e2ap7bopj75.uk.auth0.com"

clientId="dnjPPljysn6Q9zMccCz2J7gHSEKRxdzl"

authorizationParams={{

redirect\_uri: redirectUri,

audience: "https://dev-bp1s3e2ap7bopj75.uk.auth0.com/api/v2/",

scope: "read:current\_user access:posts update:current\_user\_metadata"

}}

>

<Home />

</Auth0Provider>

);

};

export default AuthenticatedApp;

## add-card.js code:

import React, { useState, useEffect } from 'react';

import { checkHostStatus } from './pingUtility';

import { updateConnectionStatus, updateName, getAllMac } from './ipMac';

import NameInput from './NameInput';

import ProgressBar from 'react-bootstrap/ProgressBar';

import 'bootstrap/dist/css/bootstrap.min.css';

import Alert from 'react-bootstrap/Alert';

const AddCard = ({ id, ip, macData, onRevokeClick, token }) => {

const [status, setStatus] = useState('Loading...');

const [macAddress, setMacAddress] = useState('');

const [name, setName] = useState('');

const [connected, setConnected] = useState('');

const [device\_type, setDevice\_type] = useState('');

const [Device\_info\_given, setDevice\_info\_given] = useState('');

const [successMessage, setSuccessMessage] = useState('');

const [alerttype, setAlerttype] = useState('');

const handleNameChange = async (newName) => {

setName(newName);

try {

const updatestatus = await updateName(ip, newName, token);

console.log(await updatestatus);

if (updatestatus === "success") {

setSuccessMessage('Name updated successfully');

setAlerttype("success");

} else {

setSuccessMessage('Error updating name');

setAlerttype("danger"); // Set alert type to "danger" for error

}

} catch (error) {

console.error('Failed to update name:', error.message);

setSuccessMessage('Error updating name');

setAlerttype("danger"); // Set alert type to "danger" for error

}

};

const handleAlertClose = () => {

setSuccessMessage('');

};

const handleRevokeClick = async () => {

try {

onRevokeClick(id);

} catch (error) {

console.error('Error revoking access:', error);

}

};

const fetchHostStatus = async () => {

try {

const hostStatus = await checkHostStatus(ip);

setStatus(hostStatus === 'Online' ? 'Online' : 'Unreachable');

} catch (error) {

console.error('Error fetching host status:', error);

setStatus('Unreachable');

}

};

const fetchConnectionStatus = async () => {

try {

if (token) {

const connectionStatus = await updateConnectionStatus(token);

setConnected(connectionStatus[ip]?.connected || false);

}

} catch (error) {

console.error('Error fetching connection status:', error);

setConnected(false);

}

};

useEffect(() => {

fetchHostStatus();

fetchConnectionStatus();

const intervalId = setInterval(() => {

fetchHostStatus();

fetchConnectionStatus();

}, 10000);

return () => clearInterval(intervalId);

}, [ip, token]);

useEffect(() => {

if (macData[ip] && macData[ip].dev) {

setMacAddress(macData[ip].mac || '');

setName(macData[ip].name || '');

setConnected(macData[ip].connected || false);

if (macData[ip].dev.device\_type) {

setDevice\_type(macData[ip].dev.device\_type);

}

if (macData[ip].dev.Device\_info\_given) {

setDevice\_info\_given(macData[ip].dev.Device\_info\_given);

}

}

}, [macData, ip]);

if (Array.isArray(Device\_info\_given) && Device\_info\_given.length > 0) {

const firstItem = Device\_info\_given[0];

if (typeof firstItem === 'string' && firstItem.substr(0, 7) === "{'bar':") {

Device\_info\_given[0] = firstItem.substr(9);

}

} else {

console.log('Device\_info\_given is not in the expected format, could still be waiting for data');

}

return (

<div className='relative text-center flex flex-col text-gray-700 bg-green-100 shadow-md bg-clip-border rounded-xl'>

<div className={`relative text-center flex flex-col text-gray-700 ${connected=== true ? 'bg-green-100' : 'bg-red-100'} shadow-md bg-clip-border rounded-xl`}>

{successMessage && (

<Alert variant={alerttype} onClose={handleAlertClose} dismissible>

{successMessage}

</Alert>

)}

<div className='p-4'>

<p>Name: {name}</p>

<p>ID: {id}</p>

<p>IP: {ip}</p>

<p>MAC: {macAddress}</p>

<div className="flex items-center justify-center space-x-2 flex-wrap">

<p>Status: {status}</p>

{(() => {

switch (status.toLowerCase()) {

case 'online':

return <span className='bg-green-500 rounded-full h-3 w-3'></span>;

default:

return <span className='bg-red-500 rounded-full h-3 w-3'></span>;

}

})()}

</div>

<div className="flex items-center justify-center space-x-2 flex-wrap">

<p>WebSocket Connection:</p>

{(() => {

switch (connected) {

case true:

return <span className='bg-green-500 rounded-full h-3 w-3'></span>;

default:

return <span className='bg-red-500 rounded-full h-3 w-3'></span>;

}

})()}

</div>

<br></br>

<p>Device Type: {device\_type}</p>

<span>

<p>Device info given:</p>

{Array.isArray(Device\_info\_given) ? (

<ProgressBar>

{Device\_info\_given.map((item, index) => {

try {

const cleanedItem = item.replace(/['":]/g, '').trim();

const [category, percentageString, variant] = cleanedItem.split(' ');

const percentage = parseInt(percentageString, 10);

if (!isNaN(percentage) && variant) {

return (

<ProgressBar

striped

variant={variant}

now={percentage}

label={`${category}: ${percentage}%`}

key={index}

/>

);

} else {

console.error(`Invalid data format at index ${index}: ${item}`);

}

} catch (error) {

console.error(`Error processing item at index ${index}:`, error, item);

}

return null;

})}

</ProgressBar>

) : (

<p>{Device\_info\_given}</p>

)}

</span>

<br></br>

<br></br>

<br></br>

</div>

<NameInput onChange={handleNameChange} />

<div className='p-4'>

<button

type="button"

onClick={handleRevokeClick}

className="focus:outline-none text-white bg-red-700 hover:bg-red-800 w-20 rounded-full m-5"

>

Revoke

</button>

</div>

</div>

</div>

);

};

const addContent = async (prevContent, onRevokeClick, token) => {

if (token) {

const macData = await getAllMac(token);

const newContent = Object.keys(macData).map((ip) => {

return {

id: ip,

jsx: () => <AddCard id={ip} ip={ip} macData={macData} onRevokeClick={onRevokeClick} token={token} />,

};

});

const filteredPrevContent = prevContent.filter((prevCard) => !newContent.some((newCard) => newCard.id === prevCard.id));

return [...filteredPrevContent, ...newContent];

}

return prevContent;

};

export { AddCard, addContent };

## ipMac.js code:

// ipMac.js

const getMac = async (authToken) => {

try {

const response = await fetch("https://hallmonitor1.tail1e215.ts.net:3002/api2/ipMacMap", {

headers: {

Authorization: authToken,

},

});

const result = await response.json();

return result;

} catch (error) {

console.error("Error fetching host status:", error);

return {};

}

};

const getAllMac = async (authToken) => {

try {

const response = await getMac(authToken);

return response; // Assuming the response contains an object with IP as keys

} catch (error) {

console.error('Error fetching MAC data:', error);

return {};

}

};

const updateConnectionStatus = async (authToken) => {

try {

const response = await fetch(`https://hallmonitor1.tail1e215.ts.net:3002/api2/ipMacMap`, {

headers: {

Authorization: authToken,

},

});

const result = await response.json();

return result; // Assuming the response contains an object with IP as keys and connection status

} catch (error) {

console.error('Error fetching connection status:', error);

return {};

}

};

const updateName = async (ipAddress, newName, authToken) => {

try {

const response = await fetch(`https://hallmonitor1.tail1e215.ts.net:3002/api2/updatename/${ipAddress}`, {

method: 'POST',

headers: {

Authorization: authToken,

'Content-Type': 'application/json'

},

body: JSON.stringify({ name: newName })

});

if (response.ok) {

console.log('Name updated successfully');

return "success";

} else {

const errorMessage = await response.text();

console.error('Failed to update name:', errorMessage || response.statusText);

return "danger";

}

} catch (error) {

console.error('Error updating name:', error.message);

return "danger";

}

};

export { getAllMac, updateConnectionStatus, updateName };

## pingUtility.js code:

// pingUtility.js

const checkHostStatus = async (ipAddress) => {

try {

const response = await fetch(`https://hallmonitor1.tail1e215.ts.net/api/ping/${ipAddress}`);

const result = await response.json();

return result.alive ? 'Online' : 'Unreachable';

} catch (error) {

console.error('Error fetching host status:', error);

return 'Unreachable';

}

};

export { checkHostStatus };

## profile.js code:

import React, { useEffect, useState } from "react";

import { useAuth0 } from "@auth0/auth0-react";

const Profile = () => {

const { user, isAuthenticated, getAccessTokenSilently } = useAuth0();

const [userMetadata, setUserMetadata] = useState(null);

const [userName, setUserName] = useState(null);

const [userPicture, setUserPicture] = useState(null);

const [userEmail, setUserEmail] = useState(null);

useEffect(() => {

const getUserMetadata = async () => {

const domain = "dev-bp1s3e2ap7bopj75.uk.auth0.com";

try {

const accessToken = await getAccessTokenSilently({

authorizationParams: {

audience: `https://${domain}/api/v2/`,

scope: "read:current\_user update:current\_user\_metadata",

},

});

const userDetailsByIdUrl = `https://${domain}/api/v2/users/${user.sub}`;

const metadataResponse = await fetch(userDetailsByIdUrl, {

headers: {

Authorization: `Bearer ${accessToken}`,

},

});

const { user\_metadata, name, picture, email} = await metadataResponse.json();

setUserMetadata(user\_metadata);

setUserName(name);

setUserPicture(picture);

setUserEmail(email);

} catch (e) {

console.log(e.message);

}

};

const getNewToken = async () => {

const newAccessToken = await getAccessTokenSilently({ ignoreCache: true });

return(newAccessToken)

}

getUserMetadata();

}, [getAccessTokenSilently, user?.sub]);

return (

isAuthenticated && (

<div>

<img src={userPicture} alt={user.name} style={{ maxWidth: "5%", height: "auto" }}/>

<h2>Username: {userName}</h2>

<p>Email: {userEmail}</p>

</div>

)

);

};

export default Profile;

## startservers.sh code:

#!/bin/bash

# Start ping server

echo "Starting ping server"

cd /var/web-apps/hallway-monitor-mainv2/ && /home/ubu/.nvm/versions/node/v18.19.0/bin/node ./servers/pingServer.js &

# Start monitor server

echo "Starting monitor server"

cd /var/web-apps/hallway-monitor-mainv2/ && /home/ubu/.nvm/versions/node/v18.19.0/bin/node ./servers/monitor.js &

# Start react server

echo "Starting react server"

cd /var/web-apps/hallway-monitor-mainv2/app/ && npm run dev

## client.py – IoT client code:

import asyncio

import websockets

import re

import uuid

import json

# File handling

CONFIG\_FILE = r'./config.json'

def get\_config\_data():

try:

with open(CONFIG\_FILE, "r") as file:

config\_data = json.load(file)

return config\_data

except FileNotFoundError:

print("Config file not found.")

return {}

except json.JSONDecodeError:

print("Error decoding JSON in config file.")

return {}

def write\_name\_to\_config(name, device\_properties):

try:

with open(CONFIG\_FILE, "w") as file:

config\_data = {

"name": name,

"Device\_prop": device\_properties

}

print(device\_properties)

json.dump(config\_data, file, indent=2)

except Exception as e:

print(f"Error writing to config file: {e}")

async def connect\_to\_server():

uri = "wss://hallmonitor1.tail1e215.ts.net:444"

mac = ':'.join(re.findall('..', '%012x' % uuid.getnode()))

print(mac)

config\_data = get\_config\_data()

name = config\_data.get("name", "YourNameHere")

# Provide default values for device properties

device\_properties = config\_data.get("Device\_prop", [])

device\_type = device\_properties[0].get("type", "Type not specified") if device\_properties else "Type not specified"

Device\_info\_given = device\_properties[0].get("info\_given", "Type not specified") if device\_properties else "Type not specified"

while True:

try:

# Establish a connection with the server

async with websockets.connect(uri) as websocket:

print("Connected to the server")

# Send MAC and name information to the server

await websocket.send(f"mymac: {mac}, myname: {name}")

# Keep the connection open indefinitely

while True:

try:

# Receive messages from the server

message = await websocket.recv()

print(f"Received from server: {message}")

# Check if the message matches a command

if message == 'sendinfo':

await websocket.send(f"info: mymac: {mac}, myname: {name}")

elif message == 'device\_prop':

await websocket.send(f"device\_type: {device\_type} Device\_info\_given: {Device\_info\_given}")

config\_data = get\_config\_data()

device\_properties = config\_data.get("Device\_prop", [])

device\_type = device\_properties[0].get("type", "Type not specified") if device\_properties else "Type not specified"

Device\_info\_given = device\_properties[0].get("info\_given", "Type not specified") if device\_properties else "Type not specified"

print(device\_type)

print(Device\_info\_given)

elif message.startswith("updatename:"):

split\_msg = message.split(":")

updated\_name = split\_msg[1]

write\_name\_to\_config(updated\_name, device\_properties)

config\_data = get\_config\_data()

name = config\_data.get("name", "YourNameHere")

except websockets.ConnectionClosed:

print("Connection to server closed")

break

except (websockets.WebSocketException, ConnectionRefusedError):

print("Unable to connect. Retrying in 5 seconds...")

await asyncio.sleep(5)

if \_\_name\_\_ == "\_\_main\_\_":

asyncio.get\_event\_loop().run\_until\_complete(connect\_to\_server())

## IoT client config.json code:

{

"name": "device #1",

"Device\_prop": [

{

"type": "air quality",

"info\_given": [

{

"bar": [

"Oxigen: 55% info",

"CO2: 30% danger",

"Other: 15% warning"

]

}

]

}

]

}

## Tailscale Access Control List:

{

"acls": [

{

"action": "accept",

"src": ["jagsblast@gmail.com"],

"dst": ["100.125.41.124:444"],

},

{

"action": "accept",

"src": ["jagsblast@github"],

"dst": ["100.125.41.124:443", "100.125.41.124:3002"],

},

],

}

Feedback and Final mark:

Achieved grade: 76%

The main aims and objectives of the project are provided. The first intro part contains some useful general information, good background information. Can be improved by adding figures, etc.

Literature review is very good and sufficiently backed up your project. It can be further improved by adding more references, and further discussion on the current practice to your project in context and establish a justification for your project work.

Really liked the detail and information put in here. Very clear and easy to understand good use of diagrams and flow charts but not sure where Magic DNS went.

Good implementation this was explained well Some screenshots were not labelled which turns them into decoration. don't assume that the reader understands why they have been included If there is no label or reference to them. It is also good practice to annotate your code for the same reason. On testing, you managed to pass every test no failures, that is a record. And it also doesn't tell you as much as the tests that fail, which you work to put right.

Good evaluation of your results and comparison with traditional methods this is a bit light and could have benefited with a bit more comparisons within Zero -Trust.

The overall aims and objectives of the report were achieved to the standard that was set in the introduction. Some interesting thoughts on future work.

Overall, a well-written report. Clear structure, format is consistent throughout the report, with good effort paying attention to the details. Very good selections of references in Harvard style. A detailed explanation is in the Appendix, including application setup on GitHub. Well done!

* Dr Bernardi Pranggono