UNIVERISTY OF GREENWICH

DEPARTMENT OF COMPUTER SCIENCE

RANSOMWARE DETECTION AND MITIGATION FOR SMART BUSINESS MANAGEMENT SYSTEM IoT NETWORKS: CONTROL AND SECURITY

THIS DISSERTATION IS PRESENTED

BY

DHIN ISLAM MD

TO

FULFIL PART OF THE REQUIREMENTS NECESSARY FOR THE DEGREE BSC HONOURS IN COMPUTER SCIENCE WITH A SPECIALIZATION IN CYBER SECURITY

April 2024

Abstract

This dissertation addresses the crucial issue of detecting and mitigating the impact of ransomware in Smart Business Management Systems (SBMS) that are enhanced by Internet of Things (IoT) networks. The growth of the Internet of Things (IoT) has significantly expanded the potential targets for cyber threats, and ransomware presents a particularly dangerous threat because of its ability to cause disruption. The research in cybersecurity specialisation aims to provide a robust and comprehensive framework for the Internet of Things (IoT) that improves operational efficiency and protects against the growing threat of ransomware attacks.

The project uses Cisco Packet Tracer to simulate an SBMS IoT network setup. It features a comprehensive security framework that includes Firewalls, Software-Defined Networking (SDN), and Intrusion Detection and Prevention Systems (IDPS). The combination of these aspects provides a defence-in-depth strategy that is crucial for identifying, reducing, and managing ransomware threats. This paper highlights the importance of flexible cybersecurity solutions that align with modern IoT networks' complexity, eventually ensuring the security and uninterrupted functioning of smart business activities.

The dissertation focuses on the strategic implementation of advanced security configurations, using Python scripting to program IoT devices and deploying simulated network situations. The paper provides a detailed analysis of how different approaches perform against ransomware attack vectors. This analysis confirms the effectiveness of the proposed security framework, adding a new viewpoint to the cyber defence strategies for SBMS IoT networks.

Acknowledgements

My deepest gratitude goes out to my supervisors for all their help, understanding, and knowledge during this project. Their advice and observations were quite helpful to me in determining the best route of action for my research.

I also want to express my sincere gratitude to my family and friends for their constant encouragement and support. Their confidence in my skills has always given me courage and inspiration.

Finally, I would want to express my gratitude to all the writers and scholars whose writings provided a foundation for this study. I am grateful to everyone who contributed to me along the way and with this project.

Declaration

I confirm that the work contained in this BSc Honours project report has been composed solely by myself and has not been accepted in any previous application for a degree. All sources of information have been specifically acknowledged and all verbatim extracts are distinguished by quotation marks.

DHIN ISLAM MD

Signed .....

Contents

List of Tables

List of Figures

”Ransomware is unique among cybercrimes because, in order for the attack to be successful, it requires the victim to become a willing accomplice after the fact.”

— James Scott

Chapter 1

INTRODUCTION

The Internet of Things (IoT) is transforming modern business practices by promoting innovative networks of interconnected devices, enhancing service models, and advancing industry sustainability. This technological evolution is particularly pronounced within Smart Business Management IoT, which has emerged as a major, attention-dominant field for its role in future advancements **Nizˇetic´ et al. 2020**. Based on the research conducted by **Gerodimos et al. 2023**, it is predicted that the IoT will see a significant expansion, resulting in an almost 300% rise in the number of connected devices. This number is expected to rise from 8.7 billion in 2020 to over 25 billion by 2030. In 2020, China held the top position worldwide in terms of IoT deployments, with an incredible number of over 3 billion devices hosted. In 2020, the retail sector included around 60% of all Internet of Things (IoT) devices. This distribution is anticipated to remain consistent over the next ten years. As the IoT continues to grow, so has the number of ransomware attacks, which have also seen a sharp increase, posing threats to individuals and businesses. This trend highlights the critical importance of organisations regularly updating their cybersecurity measures. Developing robust methods for resilience and recovery is essential to maintain uninterrupted operations in the face of these evolving threats **Humayun et al. 2021**.

Teichmann, Boticiu, and Sergi 2023 highlights in their 2022 Cyber Threat Report a startling 105% rise in ransomware occurrences in 2021. They also revealed that 37% of businesses were claimed to be grappling with ransomware infiltration that year, with the average financial impact for recovery reaching $1.85 million. The growing vulnerability of IoT-enabled smart business systems to ransomware attacks may cause significant financial disruption to services. According to **Al-Hawawreh, Sitnikova, and Aboutorab 2021**, IoT systems are likely to remain prime targets for ransomware actors.

In the era of digital technology, as the intricate network of connections becomes more complex, the growing number of IoT devices in business environments offers benefits as well as challenges. The project, named "Ransomware Detection and Mitigation for Smart Business Management System (SBMS) IoT Networks: Control and Security," focuses on the strong cybersecurity standards in smart business networks. The objective is to build an interconnected IoT system that improves operational efficiency and multi-layered security protection against the increasing risk of ransomware attacks for IoT networks.

1.1 Project Scope

The primary goal of this project is to create a Smart Business Management System (SBMS) that includes various IoT devices which connect and function together inside a secure network environment. The proposed system's architecture includes a multi-layered security framework that employs Firewalls, Software-Defined Networking (SDN), and Intrusion Detection and Prevention Systems (IDPS). These layers collaborate to identify, mitigate, and control the threats linked to ransomware and ultimately strengthen the network's defences.

An important flaw in the research is the lack of comprehensive multi-layered security in current IoT networks, which frequently exposes them to sophisticated cyber attacks. This research aims to show the effectiveness of a combined security system in a simulated network setting. The network will be elaborately built with Packet Tracer, incorporating routers, switches, and security devices configured to function smoothly. In addition, the Internet of Things (IoT) devices in this network will be programmed using Python scripts on Single-Board Computers (SBCs) and Microcontroller Units (MCUs), allowing them to carry out joint duties efficiently.

The network configuration includes establishing firewall rules and IDPS policies through command-line interactions and deploying an SDN controller to monitor and regulate network traffic. This thorough configuration is specifically developed to evaluate the network's ability against ransomware assaults and demonstrate the actual implementation and advantages of a layered security framework in the business Internet of Things (IoT) setting.

This introduction provides a foundation for a thorough review of the methods and technologies used in the project, highlighting the implementation of advanced security measures in a practical IoT network designed for optimal business management.

1.2 Chapter List

Here is a list of all the chapters within the dissertation and a brief summary of the content.

Chapter 2 Literature Review. This chapter provides an overview of existing research in IoT security, focusing specifically on network security, firewalls, IDPS, and SDN. This chapter shows how to protect IoT networks against ransomware and other cyber threats by identifying the weaknesses in current safety measures and highlighting the need for a robust, multi-layered defence.

Chapter 3 Design. This chapter explains the design of the Internet of Things network for the Smart Business Management System using flowchart diagrams, including the network topology and strategic integration of security components like firewalls, SDN and IDPS.

Chapter 4 Implementation. This chapter covers the configuration of routers and security devices, the programming of smart IoT interconnect systems using Python scripts, and practical implementation of the network design using Packet Tracer.

Chapter 5 Simulation Testing. This chapter provides a thorough report of each test's results. A table of simulation results has been created.

Chapter 6 Evaluation. This chapter explains the methods and outcomes of evaluating the network's ability to withstand ransomware attacks. It evaluates the effectiveness of the implemented security measures and the robustness of the network architecture.

”It takes 20 years to build a reputation and a few minutes of cyber-incident to ruin it.”

— Stephane Nappo

Chapter 2

LITERATURE REVIEW

2.1 Introduction

This chapter aims to provide an in-depth analysis of ransomware, including its effects on Internet of Things (IoT) networks and the evolving field of network security methods developed to mitigate these risks. This review will analyse the complex foundation of ransomware attacks and IoT and assess the efficiency of existing and emerging defence solutions in network systems. This study explores the use of Firewalls, Software-Defined Networking (SDN) and Intrusion Detection and Prevention Systems (IDPS) as important tools for securing the IoT network. The chapter will explain important ideas, discuss problems, and examine the progress made in integrating Firewall, SDN and IPS into cybersecurity protocols.

In constructing this literature review, a wide range of sources, including relevant papers and recent studies, gathered from trustworthy databases and research platforms such as IEEE Xplore, Elsevier’s ScienceDirect, the Association for Computing Machinery (ACM) Digital Library, and the digital repositories of esteemed academic institutions. This comprehensive analysis aims to establish the research within the existing knowledge structure while also creating opportunities for innovative contributions to network security in the context of IoT and ransomware.

2.2 Ransomware

The author **Ekta and Bansal 2021** defines Ransomware as a form of malware that restricts users' access to their computer systems. An extremely dangerous cyber-attack that prevents users from using their devices by locking the user's device screen or encrypting the user's files. The encrypted files stay inaccessible until a ransom is paid, and the consequences of these attacks are considered to be lasting and challenging to mitigate without the decryption key provided by the attacker. **Ekta and Bansal 2021** provides further information on the impact of ransomware, explaining that it can target both individuals and business organisations. It spreads through various methods, including email attachments, infected USB drives, and links in phishing emails. The essay highlights the dangerous nature of ransomware, which restricts access to data and exploits the anonymity offered by cryptocurrencies such as Bitcoin for ransom payments. This further complicates the task of tracing and bringing the culprits to justice. The journal's discussion emphasises the changing threat landscape, specifically highlighting the substantial growth of ransomware in recent years. Ransomware is identified as a type of malware that has become increasingly popular among hackers due to its profitability.

According to **Anghel and Racautanu 2019**, Ransomware typically functions in one of two main ways:

- **Crypto Ransomware / Encrypting Ransomware**: This type of ransomware is a popular type of malware that encrypts the files on the victim's computer. Subsequently, the user becomes incapable of accessing these data until a ransom can be paid to obtain the decryption key. This specific form of ransomware is highly damaging as it focuses on the victim's data, which could result in significant data loss if backups are unavailable.
- **Locker Ransomware**: This is a form of ransomware that does not encrypt files. Instead, it restricts the victim's access to their device, making it completely unusable. The ransomware will display an alert requesting a financial payment to reinstate the device's accessibility.

Warikoo 2023 highlights the origins of ransomware can be traced back to its early beginnings in 1989 with the AIDS Trojan. This ransomware specifically targeted the healthcare industry by spreading through infected floppy discs. As we move forward in time, the complexity of ransomware has grown. This was seen in the 2004 GPCode attack, which included phishing to distribute encryption. Subsequently, more sophisticated encryption methods were developed, such as Archievus in 2006. In 2016, Petya brought about a significant change in the field of ransomware. It utilised a strategy that efficiently immobilised entire systems by focusing on master boot records rather than individual files. The worldwide impacts of ransomware became vividly visible with the WannaCry attack in 2017, which took advantage of a security flaw with huge consequences. In the following years, there was a rise in double-extortion attacks using Maze and an increase in ransomware-as-a-service (RaaS), demonstrating ransomware's development into a fully established industry. In 2021, the emergence of LockBit 2.0 and 3.0 underlined the ongoing advancement in ransomware strategies, including bounty programmes into their strategies and reflecting a flexible and robust cybercrime ecosystem. The downfall of the infamous Conti group in 2022 and the emergence of new threat actors such as Blackbasta and Hive highlight the enduring and ever-changing nature of ransomware threats in the digital era.

2.3 Internet of Things (IoT)

The author **Laghari et al. 2021** defines the Internet of Things (IoT) as a network of physical objects, often referred to as "things," that are equipped with sensors, software, and other technologies. This includes a diverse array of applications and devices that can communicate and interact with one another via the Internet. This

connectivity leads to enhanced automation and efficiency in a variety of areas, spanning from daily household activities to industrial operations. **Laghari et al. 2021** also mentioned that the term emphasises the life-changing capacity of IoT in establishing interconnected ecosystems where devices not only collect and exchange data but also independently carry out activities based on this data, hence improving operational efficiency and user experiences across all sectors.

Elgazzar et al. 2022 presents IoT as a powerful catalyst for progress in various areas, bringing about significant developments. The authors present a model in which the IoT enables smooth communication between a large number of devices and also connects the digital and physical worlds to create intelligent surroundings. This convergence, facilitated by sensors and actuators, allows for decision-making based on data, optimising operations across several sectors such as healthcare service and urban infrastructure. In addition to highlighting the advantages, the study also brings attention to the obstacles that the IoT encounters, including security risks and interoperability issues, which have the potential to impede its progress.

According to the journal **Elgazzar et al. 2022**, tackling these problems to fully achieve IoT's potential in automating processes, increasing economic output, and improving the quality of life worldwide is essential.

2.4 Problem Domain

Given this escalating threat landscape, ransomware attacks are now targeting a broader range of devices beyond traditional computing platforms. According to **McIntosh et al. 2021**, these attacks target not just PCs and mobile phones but also smart appliances and medical equipment connected to the IoT. These devices are especially vulnerable due to their limited computing resources and security features, underscoring the importance of robust countermeasures across all interconnected devices.

Yamany et al. 2022 underscores the breakdown of ransomware infection routes, with spam accounting for 39.4%, followed by corrupted web pages at 19.8%, and vulnerabilities at 14.8%. This enumeration of attack vectors underscores the pressing importance of comprehensive security methods in IoT systems to mitigate such diverse threats.

Wheelus and Zhu 2020 emphasise the notable concerns around IoT network security and the increased vulnerability caused by various factors:

- IoT devices frequently possess limited computing and networking capabilities, which can lead to significant security vulnerabilities when integrated into IoT systems without sufficient planning and design.
- The massive growth of IoT devices presents tempting opportunities for hackers because of their huge scale and volume. Exploiting a single vulnerability can result in malware attacks.
- IoT devices are usually placed in unsecured locations, leaving them more vulnerable to attacks by attackers. Furthermore, low-cost IoT devices may not receive essential software updates and maintenance, which increases the risk of attacks.

Saeed et al. 2020 further explained the vulnerability of IoT devices to ransomware stems from their design focus on cost and power efficiency, which often comes at the expense of strong security measures. Furthermore, the complexity of IoT networks, with a vast array of interconnected devices, where each of which is a possible entry point for attackers, exacerbates the risk. This complexity of potential attack vectors highlights the challenge of detection, with studies like **Fernando, Komninos, and Chen 2020** reveals an 87% detection rate, although their dependence on decision tree techniques restricts adaptation, particularly in complicated networks. In

contrast, **Mofidi, Hounsinou, and Bloom 2023** addresses the difficulty of implementing typical ransomware detection techniques in resource-constrained IoT devices, emphasising the need for expert solutions. Current research, such as that by **Wani and Revathi 2020** introduces the innovative IoTSDN-RAN, which is a software-defined networking-based method that monitors traffic between IoT devices to detect CryptoWall ransomware. Similarly, the DAM framework by **Kapoor et al. 2021** is successful for certain ransomware types but lacks broader countermeasures against more diverse threats, such as Locker ransomware or custom-made variants. This limitation highlights the need for more versatile detection systems.

The Ransomware as a Service (RaaS) concept, discussed by **Alwashali, Abd Rahman, and Ismail 2021** indicates the simplicity with which cyberattacks can be executed, compounding the threat to global organisations. **Beaman et al. 2021** tested the antivirus effectiveness against ransomware using well-known samples, a RaaS generator, and a custom-made ransomware named AESthetic. The inability to identify tailored ransomware underlines the importance of enhanced mitigation measures, particularly in complex networks like SBMS. Lastly, **Kara and Aydos 2022** emphasizes the inadequacy of typical security tools against ransomware, proposing a multi-layered security approach. This insight has informed the project's inclusion of access control and network segmentation strategies to improve IoT network security.

According to **Abdullahi et al. 2022** in their comprehensive analysis, the rapid expansion of IoT devices and networks, integral to domains like smart homes, healthcare, and smart business systems, demands enhanced security measures to protect against growing cybersecurity threats. This clarifies the significance and timeliness of the proposed research in providing efficient security solutions for the expanding IoT network infrastructure. Therefor, the emerging trend of ransomware attacks on IoT networks, with their inherent vulnerabilities and multiple attack vectors, poses a substantial and escalating threat to the SBMS. This highlights the critical need for developing advanced, adaptable cybersecurity measures that can successfully prevent these sophisticated attacks in increasingly complex IoT systems.

2.5 Firewall

2.5.1 Introduction

Cisco Systems and Digital Equipment Corporation were the first to design network firewalls in the late 1980s, according to **Oloyede et al. 2021**. Firewalls are described as crucial tools that offer a strong defence against malicious intrusions from internet connections. **Oloyede et al. 2021** clarifies that firewalls, which can exist as hardware devices, software systems, or a combination of both, mainly allow or block network traffic depending on a set of security criteria. The purpose of this is to establish control and provide security between two networks, with the goal of safeguarding the ”INSIDE” network from potential threats emerging from the ”OUTSIDE” network. Firewalls function based on packet filtering principles, which involve controlling and filtering incoming and outgoing packets using IP addresses and port numbers. This essential role of firewalls is to protect personal and business networks from cyber attacks.

Oloyede et al. 2021 addresses the key limitations of firewalls that impact their effectiveness in protecting network security. Configuring firewalls properly requires significant time and expertise to be consistent with the network's security standards. If this complexity is not carried out correctly, it can result in security vulnerabilities. Also, in times of increased network traffic, firewalls can act as an obstacle, negatively affecting the speed and effectiveness of network activities. Furthermore, firewalls encounter difficulties when dealing with encrypted communication, as they cannot thoroughly examine the encrypted data, restricting their capacity

to identify and counteract concealed hazards inside such traffic. Finally, firewalls may face difficulties in effectively handling protocols with complex handshakes, such as FTP, without compromising a certain level of security. These constraints emphasise the necessity of implementing multi-layered security technologies that work with firewalls to improve the overall level of network protection.

2.5.2 Advancements in Firewall Technologies

Building upon the core concepts of traditional firewalls, **Mukkamala and Rajendran 2020** examines how firewalls safeguard networks against ransomware by focusing on the advancements and functionalities of advanced firewall technologies, which are next-generation firewalls (NGFWs) and cloud-based firewalls. These advanced firewalls are equipped with features such as deep packet inspection, Intrusion Prevention Systems (IPS), URL filtering, antivirus, and malware detection. By analysing the content of data packets and application-level activity, these capabilities allow them to effectively identify and block ransomware attacks. Next-generation firewalls and cloud-based firewalls have the ability to identify ransomware by detecting harmful patterns, signatures, and anomalies in network traffic by analysing the data included in the packets and observing the context of the session, these firewalls are able to detect unusual behaviours that suggest the presence of ransomware. These behaviours include unexpected encryption attempts and contact with known malicious IP addresses and domains.

Moreover, the incorporation of these firewalls into IPS and SDN settings amplifies their capacity to flexibly adjust security guidelines and protections instantly. This adaptability guarantees a stronger defence mechanism against the changing landscape of ransomware threats, offering comprehensive protection for networks, particularly IoT networks, that are vulnerable to such cyber-attacks.

2.5.3 Firewall Configuration Challenges

Given the new capabilities of Next-Generation Firewalls and cloud-based firewalls, their effectiveness can be compromised by misconfigurations and oversights, which can make the network vulnerable to attackers. **Fritts 2021** provides a comprehensive analysis of the flaws related to firewall misconfiguration in the context of network security. This study highlights that firewalls play a crucial role in network security but underlines that efficiency relies on proper configuration. Misconfigured firewalls can result in a range of weaknesses, leaving networks vulnerable to cyber risks. **Fritts 2021** outlines several configuration challenges, including rule shadowing, generalisation, and redundancy, which might compromise the efficiency of a firewall by generating conflicts or overly complex rules.

- **Rule Shadowing** refers to a situation where certain firewall rules might override or overshadow other rules, causing them to become ineffective.
- **Redundancy** refers to situations where the presence of duplicated rules in the firewall's rule base might lead to inefficiencies or conflicts.
- **Generalisation** refers to the use of overly broad rules that do not effectively secure network traffic, which might potentially enable malicious behaviour to go undetected.

The study highlights that the complexity of firewall configurations may increase the probability of errors, thus compromising the network's security. Integrating firewalls with IDPS and SDN is a crucial component of a multi-layered defence plan to mitigate these configuration challenges and enhance overall security. This approach recognises that no individual security measure is completely infallible. A multi-layered security method ensures that each layer addresses any vulnerabilities that may be left following the implementation of other

defence measures. Firewalls regulate incoming and outgoing network traffic according to pre-established rules. In contrast, IDPS and SDN actively monitor the network for indications of malicious behaviour and recognised threats. These systems effectively identify and track risks that could potentially bypass firewall protections.

2.5.4 Integrating Firewalls with Intrusion Prevention Systems (IPS)

Wu 2021 explores the improvement of network security by strategically combining Firewalls and IPS. This collaborative strategy integrates firewalls and IPS to create a stronger network security defence architecture. This approach combines firewalls' traffic filtering capabilities with IPS's dynamic detection and proactive measures, significantly enhancing a network's defence strategy, which is outline below:

- **Dynamic Threat Detection and Response**: The flawless combination of firewalls with IPS allows real-time detection and prevention of threats, enabling IPS to deal with any unusual behaviour that could potentially escape the initial firewall protections. The ability to intercept risky data packets bypassing the firewall's static rules is essential, with the IPS actively identifying and instantly eliminating such threats. In addition, this dynamic interaction enhances the network's ability to respond to new threats by constantly monitoring and updating threat databases and signatures, which are crucial for detecting emerging threats.

- **Unified Security Policy Management**: Security policy regulation is simplified by integrating firewalls and IPS into a single management framework. This unified approach allows administrators to configure and maintain security protocols more effectively, ensuring seamless operation and alignment across all network defence components. The use of centralised control is crucial in ensuring consistency of security policies across the network, hence reducing the risk of vulnerabilities that may result from separately managing firewall and IPS configurations.

- **Improved Security Posture**: The integration of firewalls and IPS improves the network's defensive capabilities, offering comprehensive and thorough protection against threats across the entire network, from the perimeter to the core. This extensive coverage not only enhances the detection of external attacks but also enables the detection and mitigation of internal threats. Furthermore, this comprehensive strategy improves the network's ability to manage advanced cyber-attacks that utilise advanced methods to bypass standard security measures.

As a result, this guarantees a higher level of security for interconnected systems, promoting a paradigm that emphasises resilience and thorough protection in the changing landscape of network security risks.

Dastres and Soori 2021 covers a thorough analysis of current network risks and different network vulnerabilities. This study also reviews the existing literature on network security threats, such as Logic attacks, Resource attacks, Passive and Active attacks, Internal and Insider attacks, and Malware attacks.

- **Logic Attacks**: This attack have the goal of exploiting any vulnerabilities in the system in order to make a profit. This involves the exploitation of software vulnerabilities, such as backdoors or vulnerabilities in the code.

- **Resource Attacks**: This attack targets the shortage or destruction of network resources. This category, which gained popularity in the 1990s, includes attacks such as Denial of Service (DoS), where the network becomes overwhelmed with an excessive number of service requests, causing it to be unable to function properly.

- **Passive Attacks**: This attack refers to unauthorised eavesdropping on the network without causing direct

damage. Because they do not alter the data, they are difficult to detect. Prevention frequently depends on strong encryption.

- **Active Attacks**: This attack refers to direct engagement with the network, such as attacking servers or deploying malware. Security systems easily detect these vulnerabilities and are frequently controlled through the use of firewalls and IPS.
- **Internal Attacks**: This attack, also known as close-in attacks, occurs when individuals gain physical access to network systems.
- **Insider Attacks**: This attack refers to malicious activities performed by individuals with authorised network access. These attacks can cause significant damage because the attackers have extensive knowledge and unrestricted access to critical information.
- **Malware Attacks**: Malware, including ransomware, refers to software designed to access and disrupt systems without the users' consent or permission. It can carry out various malicious activities, such as stealing confidential data (spyware) or demanding ransom (ransomware), which may seriously disrupt regular network operations.

The paper suggests implementing a comprehensive security approach that combines multiple security technologies at different layers. **Dastres and Soori 2021** advise the implementation of firewalls and an IDPS to provide a reliable and effective security framework. To stay aware of new threats, it is advisable to employ sophisticated security measures like the IDPS as well as perform frequent security assessments. The comprehensive review of the combination of different security layers underlines the importance of a methodical approach to network security, which is essential for protecting against a complex range of threats, including ransomware attacks.

2.5.5 Securing IoT Networks with Distributed Firewalls and Software-Defined Networks

Although the combination of firewalls and IPS enhances network security, the unique features of IoT environments require customised security approaches. Combining distributed firewalls with SDN provides a proactive method for safeguarding the various and ever-changing environment of IoT networks. **Lund, Fenzl, and Villanueva 2020** focuses on the specific challenges involved in ensuring the security of IoT networks. The study highlights the necessity for stronger network security measures in response to the limited resources of IoT devices, inconsistencies in low-level protocols, and market pressures that prioritise rapid product release over comprehensive security policies. The study also suggests using SDN to centrally manage and flexibly allocate firewall rules to strategically located fog nodes within the IoT network. The fog nodes function as localised processing units that enforce the firewall rules, enabling prompt reaction to security risks without the necessity of redirecting data to a central server. This setup minimises the time delay, improves the speed of data processing, and greatly improves the ability of security measures to handle large and diverse IoT networks.

The SDN controller in this system, which operates in the cloud, enables the network to be managed in a flexible and efficient manner by instantly updating and implementing firewall configurations. The ability to respond to changing threats and effectively manage multiple interconnections between various IoT devices and the network is vital. The distributed design of the firewalls guarantees that security measures are carried out across the entire network, hence strengthening the overall durability and ability of the IoT system to withstand attacks. This method emphasises the beneficial relationship between SDN and distributed firewall technologies, creating an effective combination that successfully tackles modern IoT networks' efficiency and security issues. The suggested solution uses SDN for flexible control and scalability and strategically places distributed firewalls for

localised threat management. This comprehensive security framework is highly suitable for IoT environments' broad and diverse nature.

2.5.6 Firewalls as Part of a Layered Security Strategy

Paga´n and Elleithy 2021 highlights the importance of implementing a layered security system to counter ransomware attacks effectively. This method incorporates a range of security measures, such as utilising well-configured firewalls, implementing SDN and IDPS, implementing reliable backup solutions, and providing comprehensive staff training. The architecture of each layer aims to intercept ransomware threats at various phases of attacks, effectively minimising the chance of intrusion and ensuring availability for quick recovery without falling to ransom demands.

Firewalls are essential in this multi-layered defence as they are the main obstacle against unauthorised access. Effectively set firewalls can prevent the spread of ransomware into the network by obstructing malicious inbound and outbound traffic, including connections to recognised malicious IP addresses and sites linked to ransomware dissemination. Implementing this proactive solution is crucial in blocking ransomware during its early phases, thereby significantly reducing the vulnerability of the network to attacks. The authors emphasise the unavoidable nature of attackers using ransomware and the significance of a proactive and all-encompassing defence plan that not only seeks to prevent attacks but also guarantees the organization's ability to recover in the case of a breach.

2.5.7 Conclusion

Firewalls are a crucial component of network security that protect against ransomware attacks. However, their effectiveness is limited when used alone. Combining them with a multi-layered security strategy significantly enhances their performance. To improve threat detection and prevention, static firewall rules are not enough. Studies show that combining firewalls with Intrusion Detection and Prevention Systems (IDPS) is the best solution. Additionally, Software-defined Networking (SDN) coupled with distributed firewalls can offer greater flexibility, scalability, and targeted threat management, which is particularly important for securing Internet of Things (IoT) networks.

2.6 Software Define Network (SDN)

2.6.1 Introduction

Software-Defined Networking (SDN) is an innovative method of designing network architecture that involves isolating the control logic of the network from the physical hardware, as highlighted by **Scott-Hayward, O'Callaghan, and Sezer 2013**. This separation enables improved programmability and dynamic administration of network resources, allowing centralised control through a software-based controller. The unique feature of SDN is its capacity to provide a comprehensive overview of the network, greatly simplifying and automating network configuration, management, and optimisation operations.

According to **Scott-Hayward, O'Callaghan, and Sezer 2013**, the architecture of SDN automatically enhances network security by enabling thorough and centralised monitoring and management. SDN can enforce network-wide security standards and adapt to emerging threats by utilising a centralised controller. This centralized viewpoint allows for real-time traffic analysis, anomaly detection, and the swift passing of security policies to mitigate potential attacks.

2.6.2 SDN for Network Security

A thorough analysis by **Maleh et al. 2023** demonstrates the value of SDN for mitigating cyber threats while improving network security. The architectural breakthrough of SDN is based on the separation of the control plane from the data plane, the centralisation of control, and the ability to be programmed. The capacity to be programmed, centralise control, and separate the control plane from the data plane constitutes the architectural innovation of SDN. Compared to traditional network topologies, these features offer a more adaptable and proactive approach to security. The implementation of centralised control optimises network administration and visibility, allowing immediate detection and mitigation of cyber threats. SDN is an effective tool for creating interconnected and responsive network security plans. Furthermore, the flexibility of SDN makes it possible to easily integrate and modify existing security systems like firewalls and IDPS right into the network architecture. The report shows how SDN may be used to deploy effective and adaptable security policies and monitor traffic in real-time.

2.6.3 Leveraging SDN for Enhanced IoT Network Security

In order to enable security measures against cyber threats, **Karmakar et al. 2020** presents an advanced security framework for IoT networks through the use of SDN. This design uses SDN's centralised control features to authenticate IoT devices and dynamically regulate network traffic, with the aim of improving the security of IoT infrastructures. By using a policy-driven approach, the framework maintains that only authenticated devices can access and interact with network services, allowing accurate access control. The main advantage of this approach is its comprehensive method of guaranteeing network flow security and authenticating devices in an IoT network. It easily combines all of these features with SDN flexibility and dynamic nature. However, the primary focus is on network security, which offers further research into the specifics of detecting and mitigating the impact of ransomware threats within the SDN-enabled IoT architecture. This paper provides a secure framework for leveraging SDN to protect IoT networks, significantly advancing IoT network security.

2.6.4 SDN-based Ransomware Detection

By introducing Software-Defined Networking into IoT network infrastructures offers a significant opportunity to improve network security against cyber-attacks. **Elsayed et al. 2023** provide a novel technique that protects IoT networks by combining complex deep learning algorithms with the configurable and centralised control of SDN. This research shows how strong deep learning techniques may be used with SDN's centralised control and network flexibility to identify and mitigate various cyber threats.

While **Elsayed et al. 2023** analyse the use of SDN in combination with deep learning, **Wani and Revathi 2020** focus on an SDN-based method for identifying and mitigating crypto-ransomware targeting IoT devices. This method detects and blocks crypto ransomware attacks by utilising Software Defined Networking (SDN). Using an SDN gateway, the IoTSDN-RAN system is designed mainly to monitor and control IoT traffic. It implements policies set up in the SDN controller to identify and stop ransomware behaviours. To find patterns of communication between ransomware and Command and Control (C&C) servers, the system extracts and analyses CoAP headers. This feature enables the system to block various attacks instantly and successfully.

Expanding on the focus of identifying crypto-ransomware attacks, **Cabaj, Gregorczyk, and Mazurczyk 2018** introduces an innovative detection approach based on analysing HTTP traffic patterns using SDN. Their method uses SDN technology to detect crypto ransomware by examining patterns in HTTP traffic. This technique specifically detects ransomware communication with C&C servers. It does this by analysing sequences of HTTP messages and their corresponding sizes. The system employs SDN's centralised control and flexibility

to actively monitor network traffic for patterns that suggest the presence of ransomware. This allows for the early identification and possible prevention of such threats. This method's effectiveness is proven by thoroughly examining traffic attributes linked to two well-known ransomware groups, CryptoWall and Locky. This analysis showcases the system's usefulness and effectiveness in real-life situations.

While previous studies focus on detection, **Alotaibi and Vassilakis 2021** examine the use of SDN to detect and actively contain self-propagating ransomware threats. The aim of this study was to examine how self-propagating ransomware can be detected and controlled in networks through the use of SDN. Their method uses a multi-module intrusion detection and prevention system to monitor network traffic for signs of ransomware activity, combining the flexible programming and centralised control of SDN. This system includes advanced techniques like as deep packet inspection, ARP scanning, and the implementation of honeypots to identify and counteract the methods used by ransomware like BadRabbit to spread.

Expanding upon the ideas of SDN-based protection, **Akbanov, Vassilakis, and Logothetis 2019** provides a detailed framework focusing on mitigating WannaCry ransomware. This framework is specifically designed to detect and mitigate the WannaCry ransomware threat by adopting SDN. By continually monitoring and analysing network traffic, this system makes use of SDN to make it easier to identify the specific communication patterns connected to WannaCry. As soon as WannaCry is identified, the framework quickly isolates it to stop it from spreading. The main method uses the OpenFlow protocol to capture suspicious network activity related to WannaCry, like unusual Server Message Block (SMB) traffic, and isolate affected network segments as soon as possible by making changes to flow table entries. After completing a detailed examination of WannaCry, the researchers discovered that the malware spreads through a variety of methods, including the use of the DoublePulsar backdoor to install malicious code on the victim's device and the EternalBlue exploit to target vulnerabilities in the SMB protocol.

2.6.5 Conclusion

In a ransomware detection study conducted on IoT networks, SDN has been shown to improve security against advanced cyber threats. **Wani and Revathi 2020**'s work, in particular, focuses on crypto ransomware and uses SDN to mitigate risks by encrypting important data. However, their studies mostly focus on crypto ransomware and do not as well address as much detail other types, like locker ransomware, suggesting the need for further research.

Additional investigation by **Alotaibi and Vassilakis 2021** shows that SDN can identify well-known ransomware families using traffic pattern analysis, but it has difficulties identifying novel or modified strains that do not follow predictable patterns. This restriction highlights how crucial it is to enhance SDN capabilities in order to lower false positives and increase detection accuracy, preventing network defences from interfering with legitimate activity. Furthermore, **Akbanov, Vassilakis, and Logothetis 2019** demonstrates the effectiveness of SDN in controlled environments, it also highlights vulnerabilities in the face of new ransomware tactics or variations. In summary, while SDN offers significant improvements in network security, its limitations require enhancing its capabilities by combining with a multi-layered defence system, including firewalls and IDPS. This integration provides a stronger security framework by combining SDN's advanced traffic analysis with IDPS's proactive threat blocking and firewalls' strict access controls. Together, these components create an effective defence strategy that can handle a wide range of cyber threats and adjust to the ever-changing landscape of network security issues.

2.7 Intrusion Detection and Prevention System (IDPS)

2.7.1 Introduction

According to **Thapa and Mailewa 2020**, Intrusion Detection and Prevention Systems (IDPS), are key tools for maintaining network security, especially in today's computer networks, where security risks are becoming more common. Intrusion Prevention Systems (IPS) and Intrusion Detection Systems (IDS) are combined to form IDPS, which are systems that include both features. In particular, IDS monitors network traffic for any potentially malicious behaviour and immediately alerts administrators. On the other hand, IPS go a step further by identifying threats and proactively taking actions to prevent them, such as blocking traffic or terminating connections. The IDPS is seen as an important strategy for maintaining the safety and security of computer systems against diverse cyber threats.

2.7.2 IDPS for IoT Network to Detect Cyber Attacks

Ramadan and Yadav 2020 have developed a new and creative hybrid IDS for IoT networks, which combines multiple advanced methodologies. Firstly, a pre-processing phase is employed to adjust and reduce the dimensionality of the data, preparing it for more efficient analysis. Next, they apply the Enhanced Shuffled Frog Leaping (ESFL) algorithm to select features, effectively identifying the most useful data. To classify potential threats, a hybrid model is used, which combines a Light Convolutional Neural Network (LCNN) with a Gated Recurrent Neural Network (GRNN). This model is specifically developed to be both lightweight and accurate. When evaluated with the NSL-KDD dataset, this system has demonstrated exceptional accuracy and detection rates. As a result, it is highly suitable for resource limited IoT networks.

Similarly, **Coulibaly 2020**'s research outlines connections between different cybersecurity methods and systems designed to identify and mitigate malware and ransomware. The article introduces IDS, which encompasses Network-Based IDS (NIDS) for monitoring entire networks and Host-Based IDS (HIDS) for individual devices, and IPS, which expands on the groundwork established by IDS. The author introduces hybrid systems, often known as IDPS, which combine the monitoring capabilities of IDS with the proactive capabilities of IPS to further strengthen security measures. Because these systems take immediate action upon detection, reducing the potential damage from cyber-attacks. The next area of study is machine learning (ML) technology, which enhances these systems' predictive capacities and enables greater adaptation to new threats. In order to ensure that these systems continue to function effectively as cyber threat landscapes change, the discussion ends by highlighting the significance of ongoing evaluation and updates. This smooth transition from simple detection to complex prevention emphasises the document's multilayered approach to cybersecurity.

2.7.3 IDPS Limitations and Challenges

Similar to the studies by **Ramadan and Yadav 2020** and **Coulibaly 2020** that focus on IDPS's hybrid detection, **Beigh, Bashir, and Chahcoo 2013** provides a comprehensive analysis of the existing barriers and restrictions currently affecting the IDPS field. An important issue that has been underlined is the dependency on incomplete or outdated data sources for the testing and training of IDPS. The lack of this capability greatly restricts the ability of the systems to detect and respond to upcoming cyber threats. Therefore, it is necessary to provide updated and comprehensive data sets that accurately represent current network environments and methods of attack. Moreover, the efficiency and rapidity of existing detection algorithms are being tested. Algorithms that can quickly adjust to and efficiently manage the constantly changing patterns of malicious threats are urgently required. These developments would greatly improve the rate and dependability of IDPS in real-life situations.

The paper also discusses the difficulties presented by the addition of various data types to present-day networks. Given the varied data types often found in networks, it is crucial for IDPS to effectively manage and integrate these formats to ensure strong security measures throughout all levels of network interaction. Another significant challenge is the platform dependency of existing IDPS solutions, which often necessitates specific system requirements for implementation. This limitation restricts the wide application of IDPS in various IT infrastructures. The proposed solution is a shift towards a more modular design in IDPS architectures, allowing for updates to be implemented without disrupting the system's operational performance.

2.7.4 Conclusion

Implementing the suggested improvements is crucial to overcoming these challenges. However, it is important to note that IDPS alone is insufficient to address all the issues. A multi-layered security approach is necessary to strengthen the entire defence system, ensuring the presence of various safeguards to detect any threats that may bypass initial detection layers.

”60% of data breaches are caused by a failure to patch. If you correct that, you’ve eliminated 60% of breaches. And I didn’t even have to say AI or Blockchain! See how that works?”

— Ricardo Lafosse

Chapter 3

DESIGN

This chapter provides the Smart Business Management System (SBMS) foundation, starting with an in-depth design presentation of the network topology. This chapter outlines the interconnected structure of the SBMS network, providing detailed information about its infrastructure and the crucial connections between its components. The detailed flowchart diagrams consistently show the connections among network components and the strategic measures adopted to prevent ransomware attacks. These diagrams describe the order of events from the initial identification of a threat to its final resolution, highlighting the crucial role played by each component of the security structure.

3.1 Network Topology

The design of the core network infrastructure is crucial for ensuring uninterrupted and secure communication throughout the SBMS’s network. The core of this architecture consists of three routers connected to each other using serial cables, creating a wide-area network (WAN) topology. This setup guarantees accurate data

transmission across network segments that are spread out geographically.

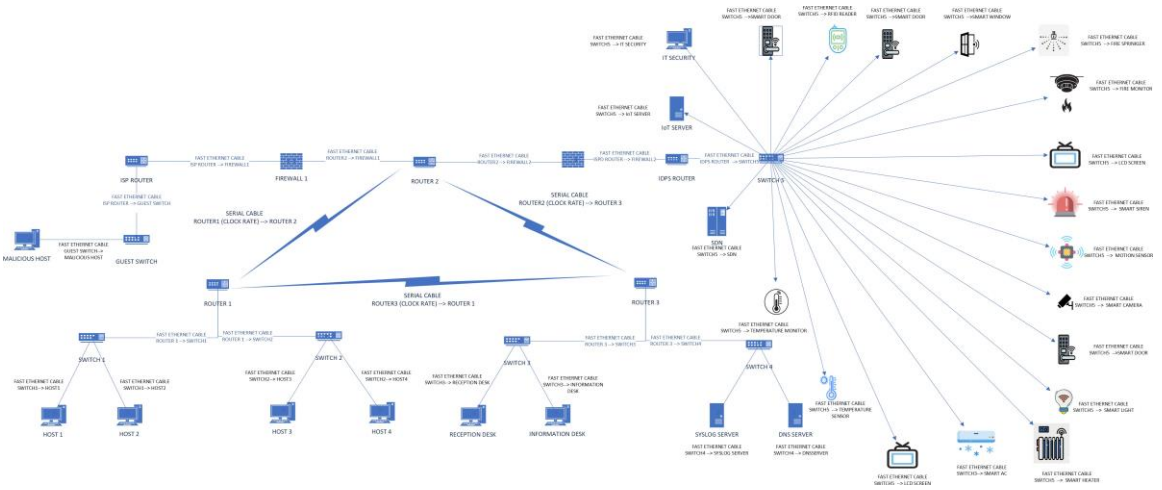


Figure 3.1: Network Topology Diagram

Figure 3.1 shows that Router 1 serves as the Data Circuit-terminating Equipment (DCE) for Router 2. On the other hand, Router 1 also functions as a Data Terminal Equipment (DTE) when connecting with Router 3, showing its ability to perform two different roles in the network. Router 2 performs the role of DCE for Router 3. Alternatively, it functions as a DTE to Router 1. Router 3 operates as the DCE for Router 1. As the DCE for Router 2's connection, it receives and uses the clock rate provided by Router 2.

The Internet Service Provider (ISP) router, a crucial component in the network structure, has been strategically positioned before Firewall 1. This placement is crucial because it serves as the point where the external internet environment and the SBMS's internal network meet, serving as the main entry and exit point for all traffic entering and leaving the network.

Switches play a vital role in a well-designed network by acting as the primary hubs that enable communication between devices and the wider network architecture. These switches not only allow device communication but also permit the separation of the network into controllable and secure domains. A group of switches is strategically placed to create separate operational zones, each dedicated to specific departmental tasks and services inside the SBMS.

The network topology's security framework is enhanced by strategically positioned devices that protect and manage the data flow. The devices include Firewalls, an Intrusion Detection and Prevention System (IDPS) router, and a Software-Defined Networking (SDN) controller.

Firewall 1 has been planned between the ISP router and the internal network to act as the main security barrier for incoming and outgoing network traffic. The device's connection to the ISP router and Router 2 highlights its responsibility to protect the network's perimeter. Firewall 2 is positioned on the opposite side of Router 2 and serves as an additional protective barrier by connecting to the IDPS router. The IDPS Router is positioned between Firewall 2 and Switch 5, acting as a downstream connection from the firewall and an upstream connection to the switch. Additionally, the SDN Controller is connected to Switch 5

3.2 IoT Interconnection Topology

Figure 3.2 provides an in-depth visualisation of the IoT topology, showing the interconnection of different smart devices and their integration into Single-Board Computers (SBC) and MicroController Units (MCU). The arrangement highlights the complex IoT device network that plays a crucial role in automating and monitoring the Smart Business Management System (SBMS).

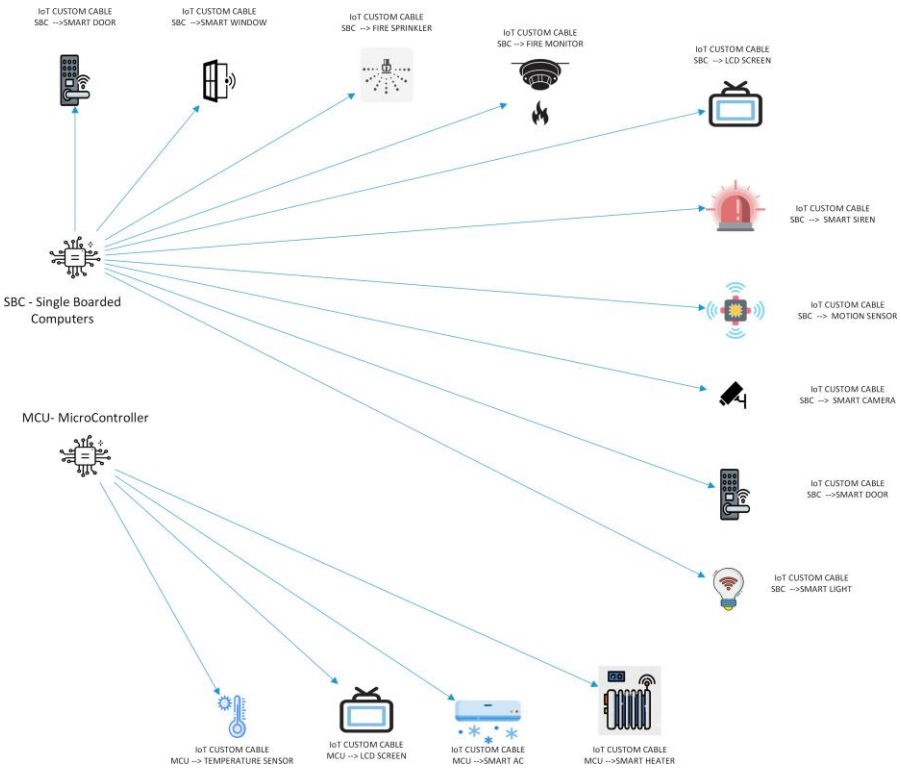


Figure 3.2: IoT Interconnected Topology Diagram

The topology prioritises the SBC as the central computing platform, giving the required processing capability for a range of IoT interconnected devices. The SBC is programmed to react to crucial fire-related incidents. When a fire is detected, the system immediately activates sprinklers, unlocks doors, and opens windows. Meanwhile, the LCD screen shows visual alerts and the siren generates a sound to signal quick evacuation. The SBC also includes a motion sensor feature, which advances security and improves operating efficiency. The motion sensor sets off a series of automated actions when movement is detected.

The MCU, operating parallel with the SBC, is fitted with a temperature sensor that actively measures the building's surrounding environmental conditions. This sensor plays a key part in the environmental control system, continuously monitoring the temperature to ensure a normal indoor atmosphere.

3.2.1 Smart Fire Alert System

The UML Diagram in **Figure 3.3** represents the chain of operations that the fire alert system will execute in response to possible fire incidents. This diagram functions as a detailed plan for the planned fire safety procedure within a building.

The process starts with system initialization, which involves configuring the required General Purpose Input/Output (GPIO) modes for the connected devices. Subsequently, the system initiates a monitoring loop, which involves the constant collection of data from the fire sensor. Upon detection of a fire, the system begins fire response processes, which involve activating alarms, sprinkler systems, and evacuation alerts. The system will stop monitoring if no sign of fire continues beyond 10 seconds.

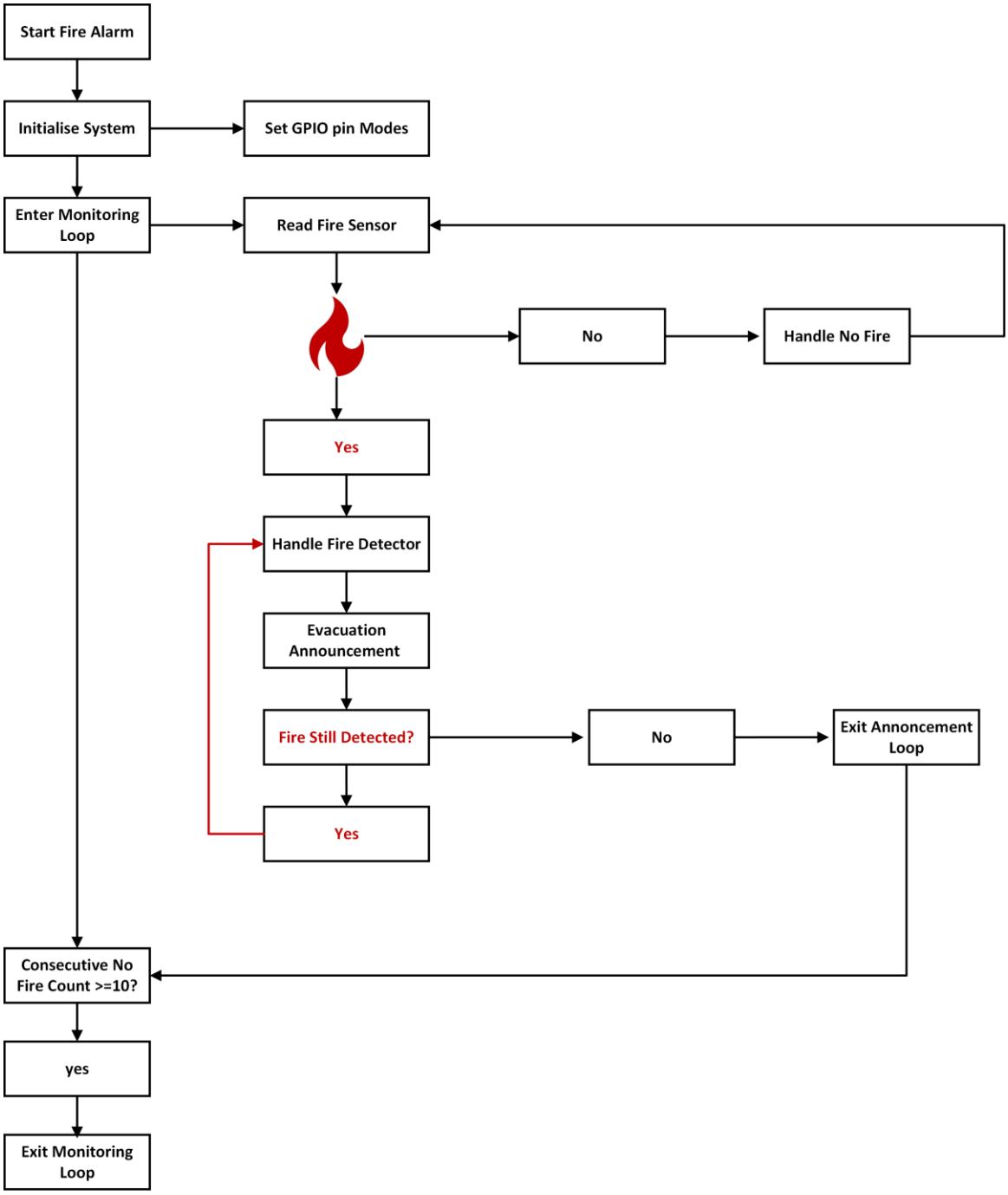


Figure 3.3: Smart Fire Alert System Flowchart

3.2.2 Smart Motion Detection System

The below diagram in **Figure 3.4** outlines the planned actions for a smart motion detection system. This system will use motion-detecting technology to automate door, light and camera operations.

At first, the system configures the GPIO pins for the motion sensor, camera, door mechanism, and light. After being initialised, the programme begins a loop specifically designed for detecting motion. When motion is detected, it initiates related actions such as activating the camera, opening the door, or turning on the light. In the absence of any observed motion, the system will persistently monitor for motion. The system will end the detection loop once the system detects 10 consecutive instances without any motion.

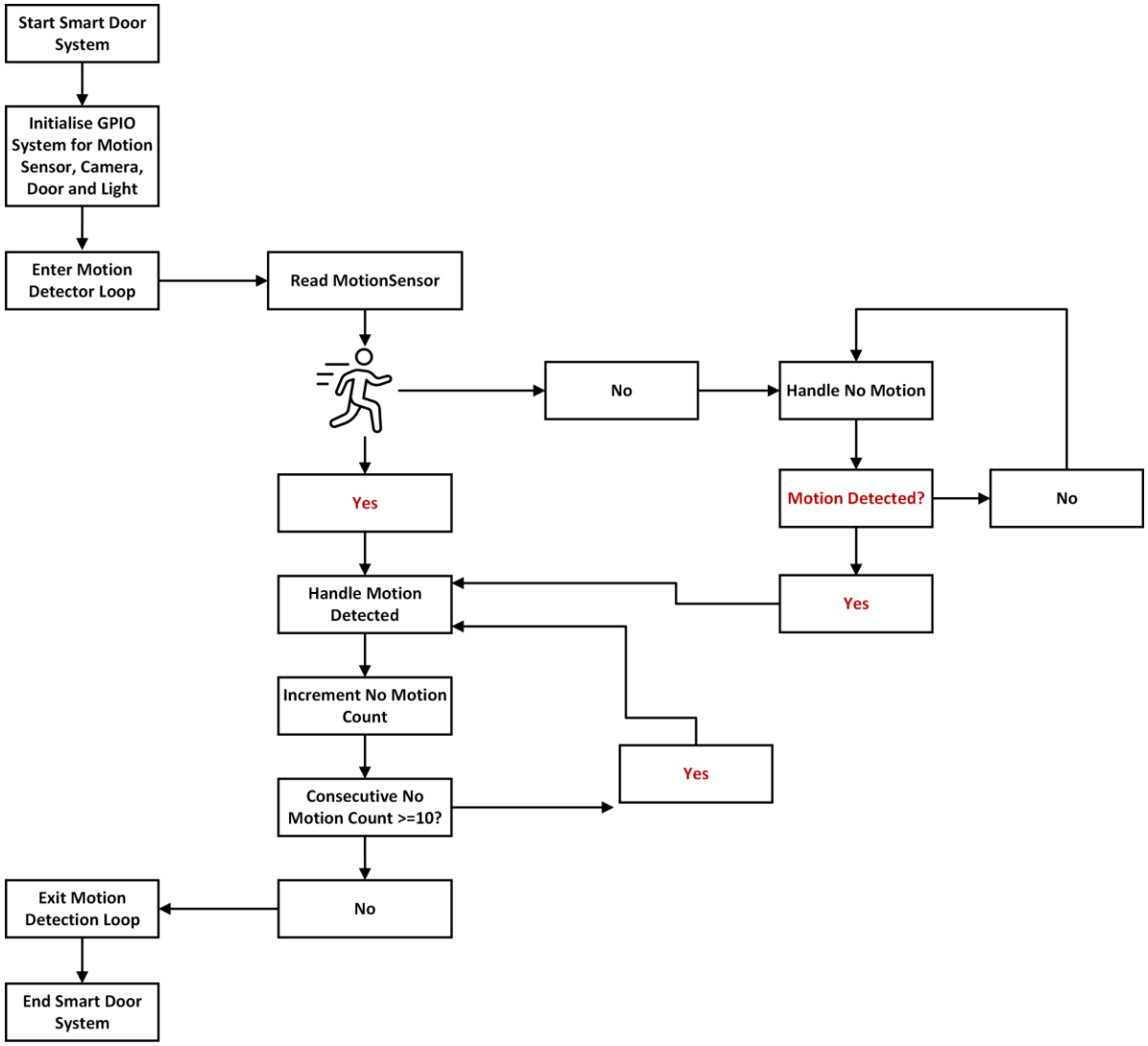


Figure 3.4: Smart Motion Detection System Flowchart

3.2.3 Smart Temperature Control System

Figure 3.5 explains the planned operation of a smart temperature control system. This automated system will determine the temperature of the SBMS's building and activate heating or cooling devices in order to maintain ideal conditions.

The procedure commences by initialising the GPIO configuration for the air conditioning (AC), heater, and

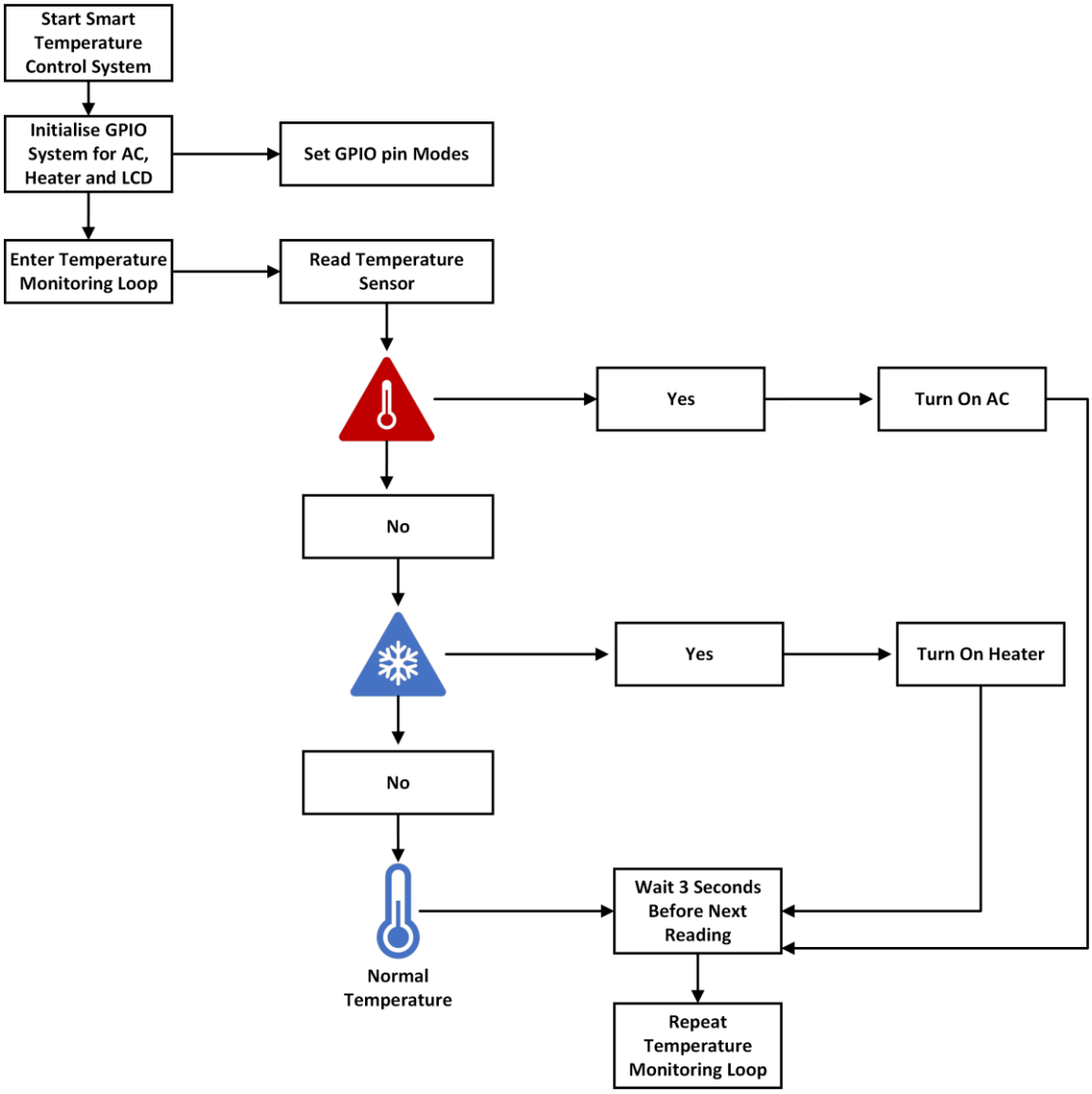


Figure 3.5: Smart Temperature Control System Flowchart

LCD display. Once the GPIO pin modes are configured, the system starts a loop to monitor the temperature. The appliance uses the temperature sensor to measure the temperature and then decides if it is too high or low. If the temperature is too high, it activates the air conditioning system, and if the temperature is too low, it triggers the heater. If the temperature falls between high and low, the system will keep its current state in order to maintain a constant temperature. This process will be iterative, with a 3-second wait between readings to maintain maximum effectiveness. The loop will iterate, consistently changing the room's temperature as required to maintain normal temperature.

3.2.4 Smart ID Card Reader System

The UML Diagram in **Figure 3.6** shows the early design of the RFID Security System and explains the sequence of operations for a security system that uses Radio-Frequency Identification (RFID) for access control.

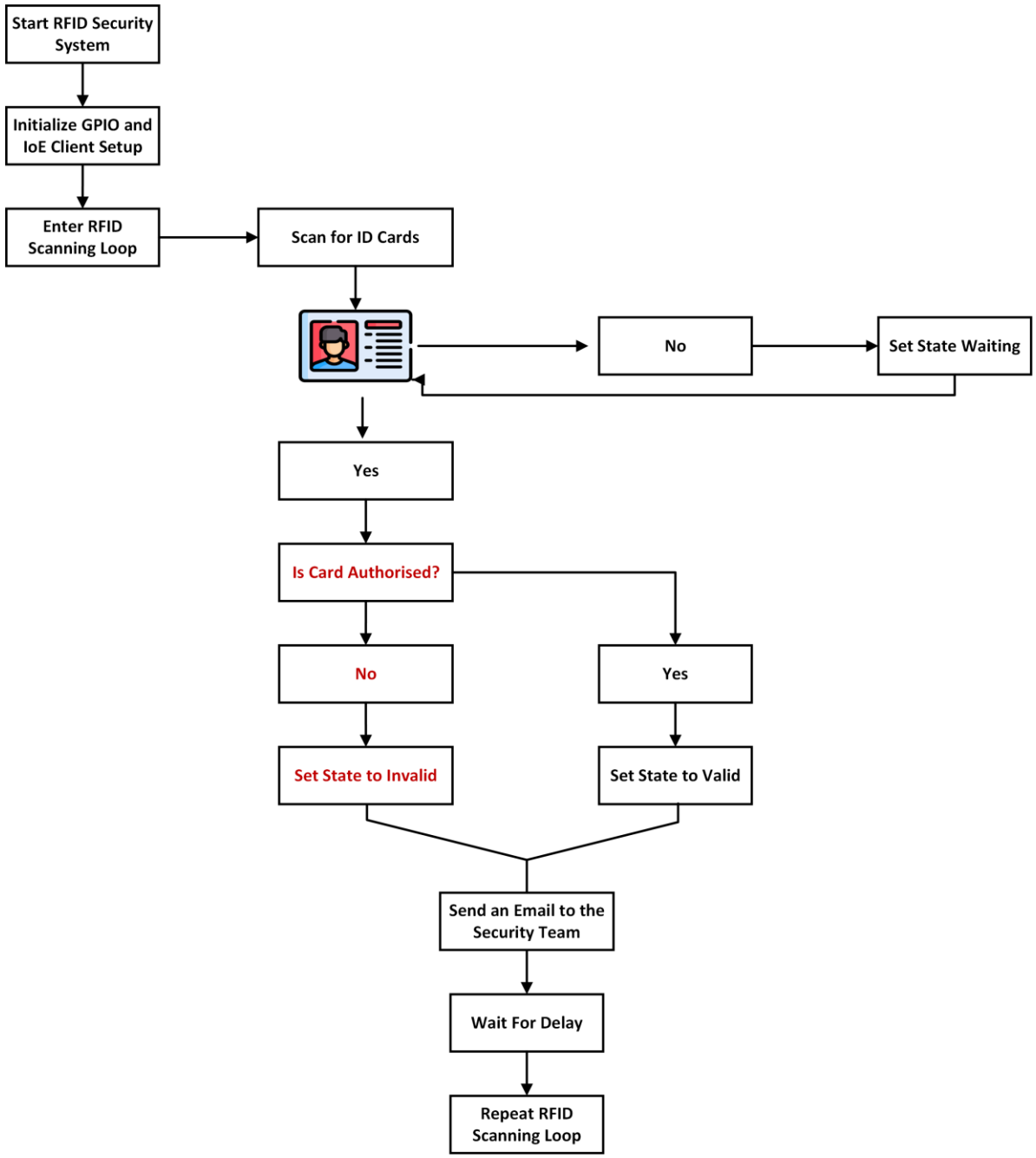


Figure 3.6: RFID Security System Flowchart

The entire procedure begins by beginning the GPIO of the system and configuring the Internet of Everything (IoE) client. The system subsequently performs a constant scan for ID cards. If no card is discovered, it remains in a state of waiting. Upon detecting a card, the system checks its authorisation status. Upon authorisation, the state gets recognised as valid, or if the card is not authorised, the system will declare its condition as invalid and dispatch an email to the security team. After either event, there is a brief delay before the system returns the RFID scanning loop to monitor for new ID cards continuously.

3.3 Ransomware Payload

In the ever-changing world of network security, it is essential to understand the various sources of risks. **Figure 3.7** offers a clear representation of the interaction between a malicious host and an IoT device door in a network. This interaction is a systematic process in which an unauthorised individual tries to gain control of connected devices, potentially to disrupt, exploit, or breach the integrity of the networked environment.

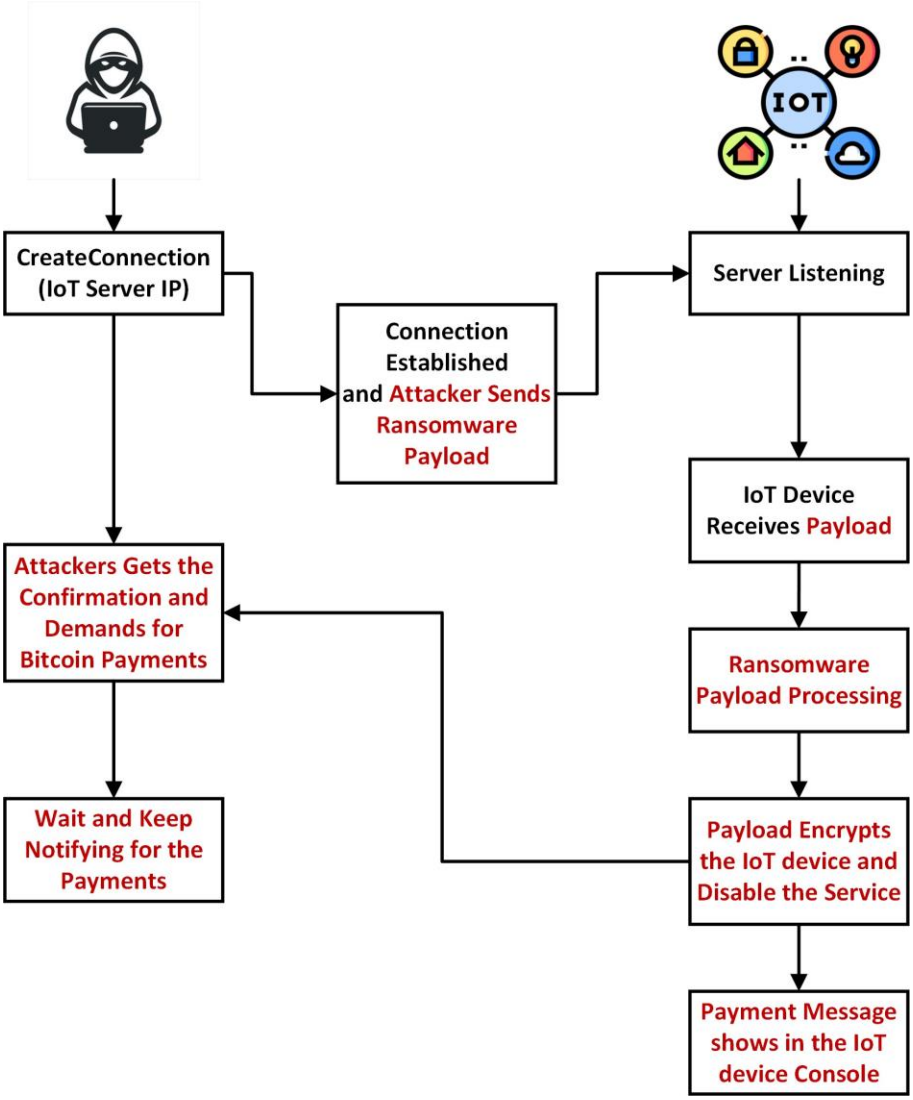


Figure 3.7: Ransomware Payload Flowchart - 'Malicious Host to IoT Device'

Figure 3.7 demonstrates the exchanges between a malicious host and smart door in the context of network communication. The malicious host will create a TCP client object. After being created, the client will establish a connection to the IoT device, using the connect(ioserverIP) method. Once the IoT device acknowledges the connection through the onConnectionChange callback, the malicious host will transmit a Protocol Data Unit (PDU) to the door. This PDU will include a command called "TRIGGER_RANSOMWARE".

3.4 Firewall

Within the proposed security protocol, Firewall will serve as the first line of defence, constantly inspecting incoming communications for any signs of ransomware. The diagram below, **Figure 3.8** and **Figure 3.9** present a logical visualisation of the processes that the Firewall will execute when it is activated and installed within the security perimeter.

3.4.1 Firewall 1

Figure 3.8 shows that the firewall task will start when a malicious host attempts to transmit traffic containing a ransomware payload specifically aimed at breaching the internal network's security measures. After receiving the data packets, Firewall 1 will begin the comprehensive inspection process. This task will require an in-depth inspection of the packet, comparing signatures with known threats to identify the characteristics of the payload.

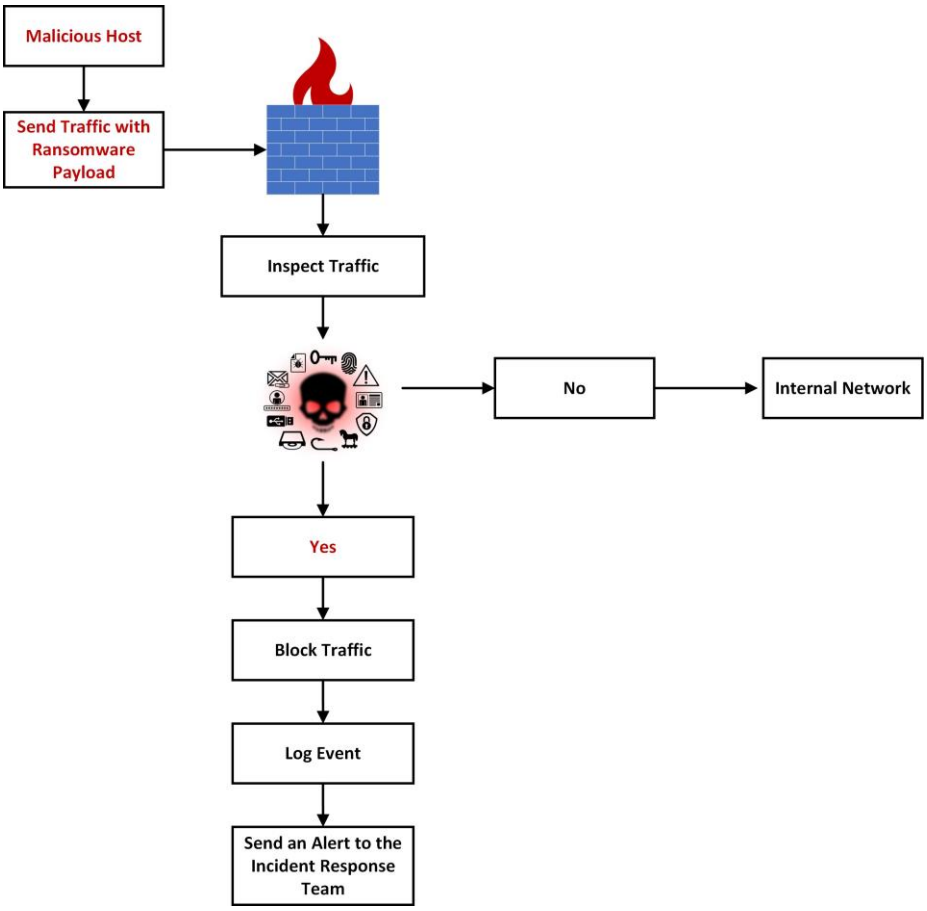


Figure 3.8: Firewall 1 Flowchart

A crucial decision point will be reached when Firewall 1 evaluates whether the payload displays typical features of a ransomware attack. If the inspection determines that there is no malicious activity, the traffic will be permitted to enter the internal network without any challenges, which guarantees continuous business operations. If the payload appears to be malicious, Firewall 1 will implement a sequence of defensive tactics. The primary action will be to block the flow of traffic, preventing malicious data from entering or compromising any part of the internal network. After implementing this passive defence method, Firewall 1 will log the occurrence in its security incident records. This recording will lead to a crucial chain of evidence that will be essential for future

forensic investigation, following regulatory standards, and the improvement of threat intelligence.

After recording the event, Firewall 1 will immediately send an alert to the Incident Response team. The message will usually contain information about the incident, such as the particular ransomware signature detected, the traffic's source, and the exact time when it occurred. These specific pieces of information will allow the security team to quickly take additional steps to investigate to fix the problem and remove the malicious host.

3.4.2 Firewall 2

The diagram, **Figure 3.9** displays an Insider Threat sending out traffic that is suspected to contain a ransomware payload. The traffic, which may contain malicious code, is targeted towards IT security and IoT network systems. Firewall 2 will function as the most important checkpoint.

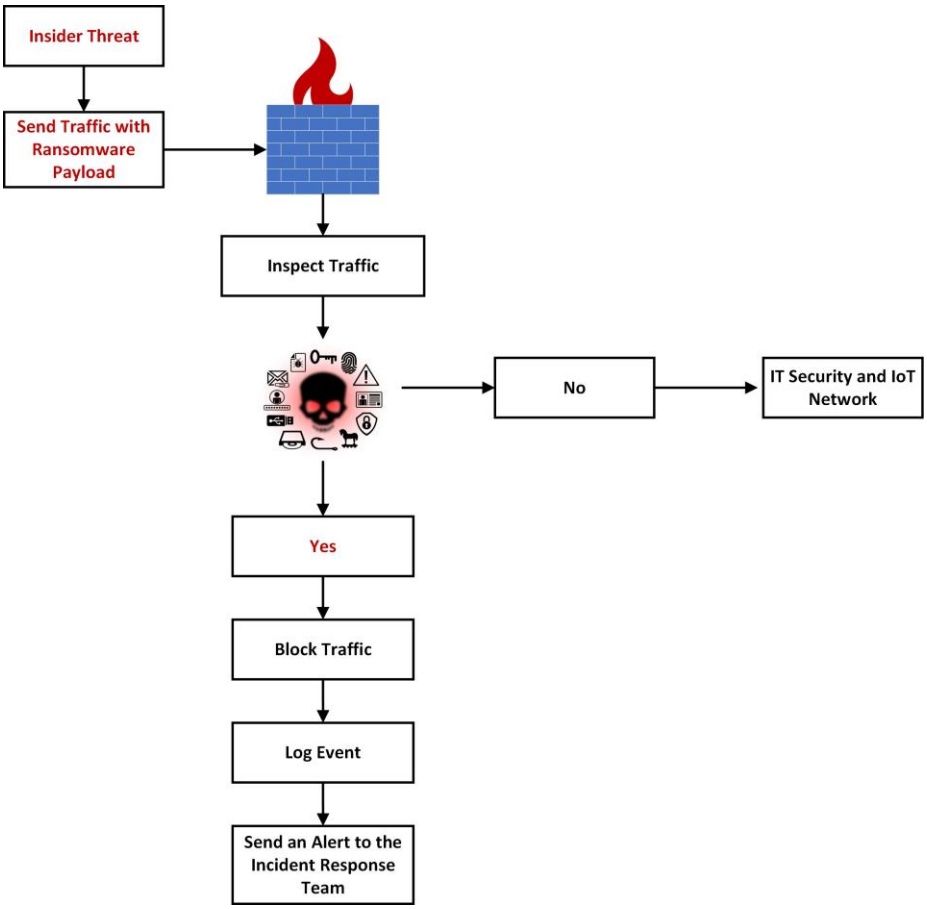


Figure 3.9: Firewall 2 Flowchart

Like Firewall 1, Firewall 2 will capture this traffic to examine it. When the payload is recognised as malicious, Firewall 2 blocks the communication, records the security event, and notifies the security team. If the payload is determined to be non-malicious, it is permitted to pass through to the IT security and IoT network.

3.5 Software-Define Networking (SDN)

The current design stage is focused on building the SDN Controller, which will function as the network's central nervous system for monitoring its operations. The system's main function will be to continuously analyse the data flow within the SBMS's Network, with a special focus on IT security and IoT networks, actively looking for behaviours of potentially malicious activities.

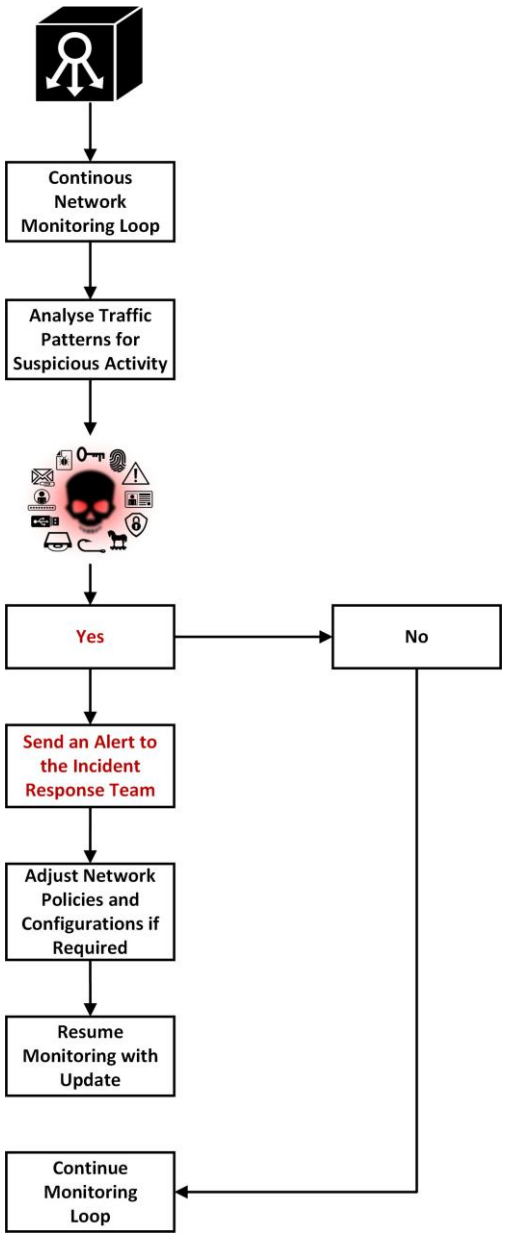


Figure 3.10: SDN Controller Flowchart

Figure 3.10 presents that activation of the SDN Controller will initiate a continuous network monitoring loop, carefully examining traffic for any indications of anomalies. An automated alert will instantly notify the IT security staff if suspicious actions are detected. The controller's analytical ability will identify between regular and malicious network activity, maintaining network stability while minimising false identifications. In response to true threats, it will act swiftly to adjust network policies and configurations to strengthen the defences.

3.5.1 Service Ticket

Within a Software-Defined Networking (SDN) environment, a service ticket serves as a key security token that provides authorised access to the functionality of the SDN controller. The service ticket guarantees that only authorised users or systems can modify network configurations or request information regarding the network's status.

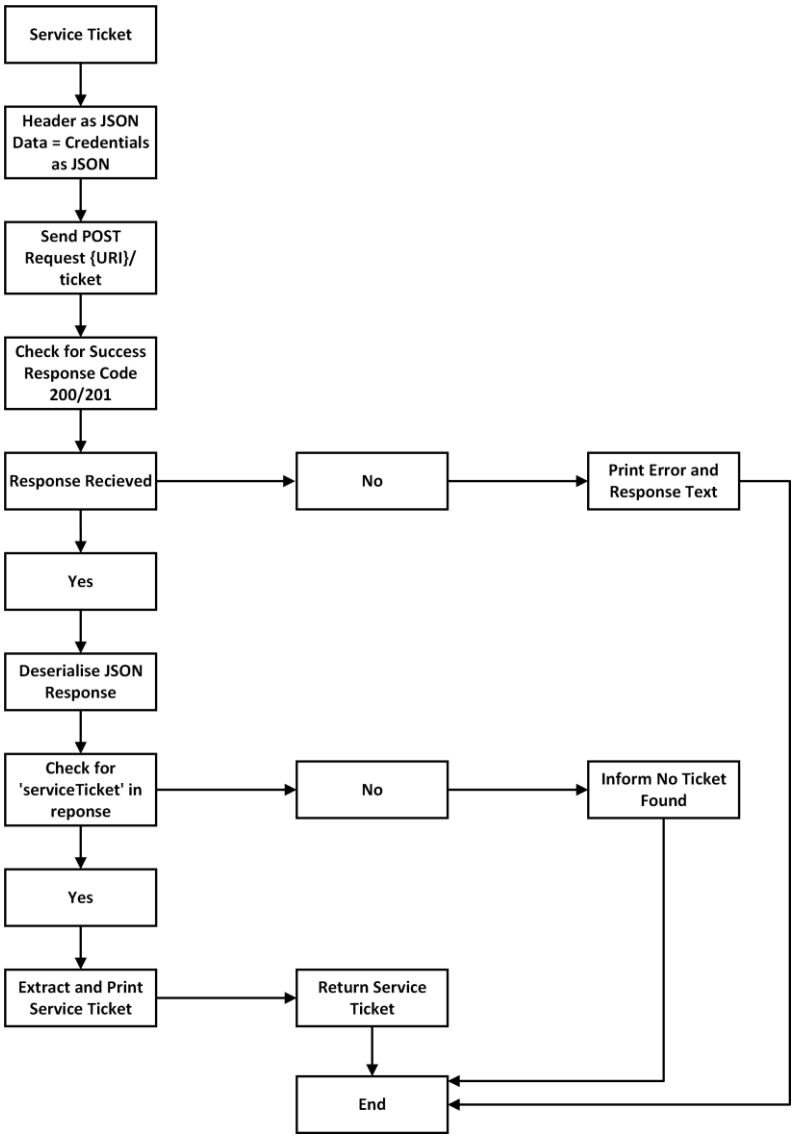


Figure 3.11: Flowchart of Obtaining Service Ticket using Python Script for SDN

Figure 3.11 demonstrates the step-by-step process of obtaining a service ticket from the SDN Controller in the Python script. At first, the system will convert the headers and the login credentials into a JSON structure. Then, these credentials will be used to send a POST request to the SDN's authentication service. After receiving a response, the system will confirm the task's success by reviewing the response code. If the code indicates success, the JSON response will be deserialised to retrieve the 'serviceTicket'. Alternatively, the system will record the error and print it. Upon locating the 'serviceTicket' in the response, the system will display and provide the ticket for further tasks. If no ticket is detected, the system will immediately notify the user that no ticket has been retrieved.

3.5.2 Network Devices and Host

The diagram **Figure 3.12** and **Figure 3.13** describes the step-by-step process for an SDN Controller using a Service Ticket to verify and obtain information about network devices and hosts. The process flow will include the crucial stages, from opening a session with JSON headers to extracting and listing network devices and host information.

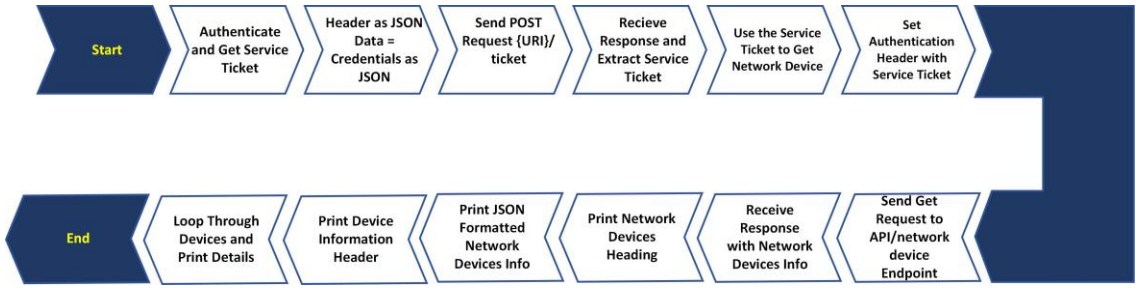


Figure 3.12: Flowchart of obtaining Network Devices Information Using Service Ticket - Python Script

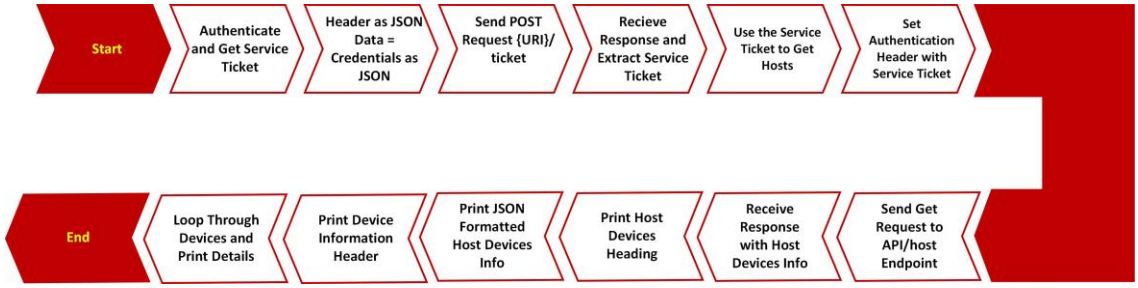


Figure 3.13: Flowchart of obtaining Host Information Using Service Ticket - Python Script

As previously mentioned, the same process can be used to obtain a Service Ticket. After receiving the service ticket, the system will send a GET request to the network-device and host API endpoint in order to obtain a list of network devices and hosts. The response will provide a list of devices, which will then be delivered in a well-organised JSON format for inspection. The process will progress by printing a section header highlighting the change to presenting device details. A systematic iteration will be conducted, analysing each network device and host in detail and displaying relevant information such as hostname, serial numbers, and software versions. This organised procedure will come to an end with the authentication and retrieval of data.

3.6 Intrusion Detection and Prevention System (IDPS)

The **Figure 3.14** is the design of the network's IDPS, outlining a list of steps the IDPS router will take to prevent ransomware attacks. It provides a visual representation that helps in understanding the measures taken to protect against malicious activities attacking the IT Security and IoT network departments.

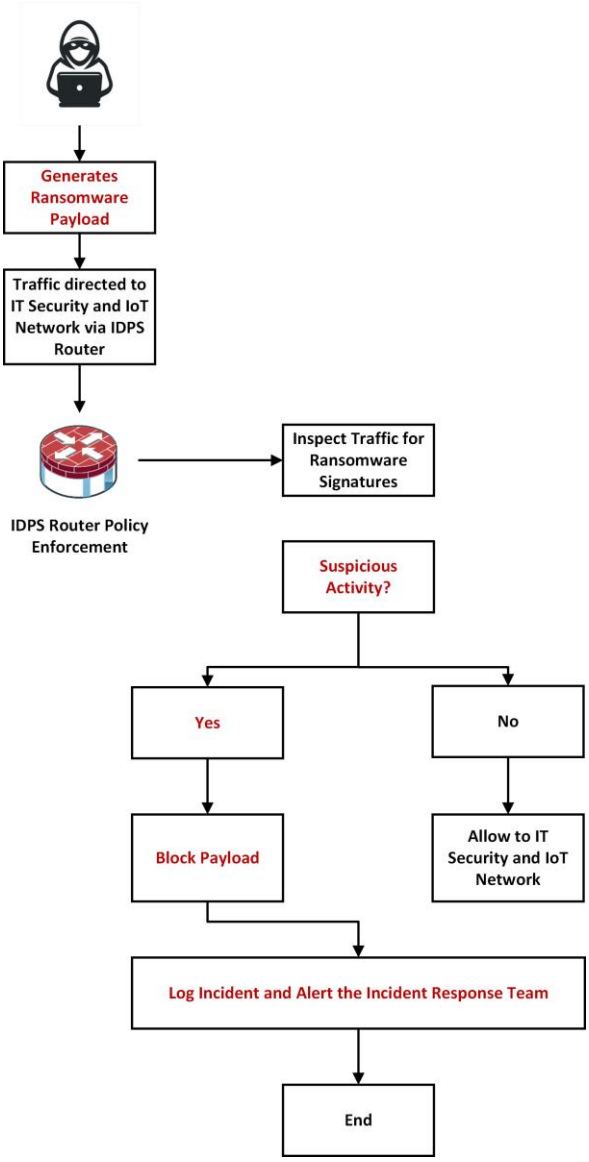


Figure 3.14: Intrusion Detection and Prevention System Flowchart

The process will start when a malicious host or insider threat creates and attempts to send out a ransomware payload. The payload is routed to the IDPS router, which serves as the network's detection and prevention system, monitoring and analysing network traffic. Once the ransomware signatures are detected, the router implements restrictions to prevent the malicious payload from entering the network. It additionally logs the incident for documentation purposes and immediately alerts the incident response team to take necessary action. In situations where traffic is not considered a risk, it is permitted to travel without interruption to its targeted destination within the IT Security and IoT network.

”A secure system is one that does what it is supposed to.”

— Eugene Spafford

Chapter 4

IMPLEMENTATION

The implementation chapter in this dissertation shows real-world implementation of theoretical knowledge through the use of Cisco Packet Tracer, a powerful network simulation tool that enables the systematic building of SBMS IoT networks. The simulation environment plays a crucial role when evaluating the success of the broad cybersecurity measures suggested in this research. The platform offers an interactive environment where different network components are properly set up, allowing for an in-depth analysis of the system’s ability to withstand advanced ransomware attacks. The practical knowledge obtained from these simulations is crucial in strengthening the suggested multi-layered security structure, guaranteeing the integrity and resilience of the SBMS against new cyber threats.

4.1 Smart Business Management System’s IoT Network

The implementation chapter focuses on the network topology system, an essential part of the SBMS. This stage is defined by an evolution from design planning to practical implementation, developing ideas into an interconnected and functional network. The **Figure 4.1** presents the actual network structure, which has evolved directly from the design drafts into a functional network architecture that supports the whole SBMS network.

The network is equipped with Cisco ISR 4331 routers that perform many functions, such as serving as the ISP ROUTER, IDSP ROUTER, ROUTER1, ROUTER2, and ROUTER3. Cisco 3560-24PS switches, which are layer 3 switches, are used for their switching capabilities. Cisco ASA 5506 firewalls enhance security. The SDN Controller is managed through the common Network Controller, ensuring secure network operations. SBC and MCU are combined in the system to enable IoT capabilities specifically designed for PT.

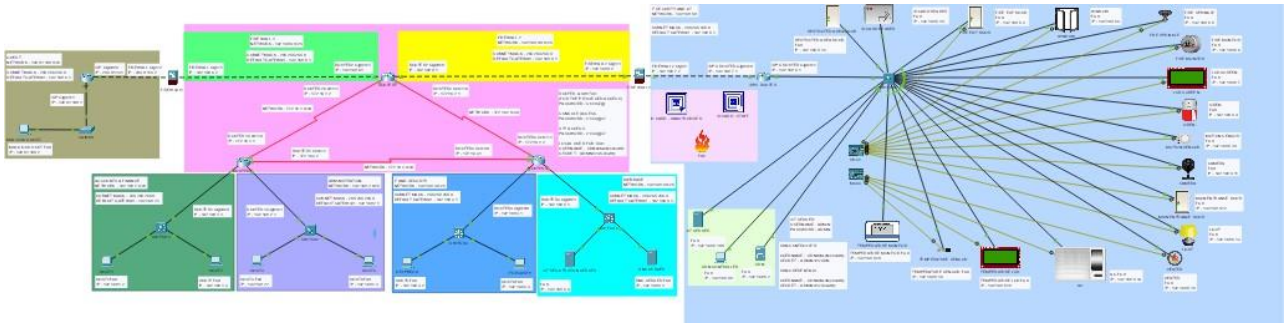


Figure 4.1: Smart Business Management System's IoT Network

The network topology contains three main routers, connected by serial cables for reliable communication. Every connection between the routers is assigned with a unique subnet, which improves network administration and segmentation. The following details give an in-depth breakdown of the connections and configurations:

Router 1 to Router 2 Connection - Network: 172.16.1.0/30 and Subnet Mask: 255.255.255.252

Router 2 to Router 3 Connection - Network: 172.16.2.0/30 and Subnet Mask: 255.255.255.252

Router 3 to Router 1 Connection - Network: 172.16.3.0/30 and Subnet Mask: 255.255.255.252

Connection	Router 1 IP	Router 2 IP	Router 3 IP
Router 1 to Router 2	Serial0/1/0 - 172.16.1.1	Serial0/1/1 - 172.16.1.2	N/A
Router 2 to Router 3	N/A	Serial0/1/0 - 172.16.2.1	Serial0/1/1 - 172.16.2.2
Router 3 to Router 1	Serial0/1/1 - 172.16.3.2	N/A	Serial0/1/0 - 172.16.3.1

Table 4.1: Routers IP Address Table

Note: In the **Table 4.1**, Serial0/1/0 has assigned as DCE end with clock rate set and Serial0/1/1 has assigned as DTE end with no clock rate

The connection between routers and switches within the network topology is shown in **Table 4.2**:

Router	Router Interface	Connection Type	Switch	Switch Port	Department
Router 1	Gi0/0/0	Copper Straight Through	Switch1	Fa0/1	Accounts and Finance
Router 1	Gi0/0/0	Copper Straight Through	Switch2	Fa0/1	Administration
Router 3	Gi0/0/0	Copper Straight Through	Switch3	Fa0/1	Reception
Router 3	Gi0/0/0	Copper Straight Through	Switch4	Fa0/1	Database
IDPS Router	Gi0/0/0	Copper Straight Through	Switch5	Fa0/1	IT Security and IoT
ISP Router	Gi0/0/0	Copper Straight Through	Guest Switch	Fa0/1	Guest Network

Table 4.2: The connection between Routers - Switches

The network topology consists of 10 separate networks, each assigned to a certain department within the SBMS. Segmentation improves both the security and efficiency of managing networks throughout the system. Below **Table 4.3** represents the IP table for the network topology:

Network Description	Network ID	Subnet Mask	Device IPs	Device Description	Default Gateway
Accounts and Finance	192.168.1.0	255.255.255.0	192.168.1.1	ROUTER1	192.168.1.1
Accounts and Finance	192.168.1.0	255.255.255.0	192.168.1.2	HOST1	192.168.1.1
Accounts and Finance	192.168.1.0	255.255.255.0	192.168.1.3	HOST2	192.168.1.1
Administration	192.168.2.0	255.255.255.0	192.168.2.1	ROUTER1	192.168.2.1
Administration	192.168.2.0	255.255.255.0	192.168.2.2	HOST3	192.168.2.1
Administration	192.168.2.0	255.255.255.0	192.168.2.3	HOST4	192.168.2.1
Reception	192.168.3.0	255.255.255.0	192.168.3.1	ROUTER3	192.168.3.1
Reception	192.168.3.0	255.255.255.0	192.168.3.2	RECEPTION DESK	192.168.3.1
Reception	192.168.3.0	255.255.255.0	192.168.3.3	INFORMATION DESK	192.168.3.1
Database	192.168.4.0	255.255.255.0	192.168.4.1	ROUTER3	192.168.4.1
Database	192.168.4.0	255.255.255.0	192.168.4.2	DNS SERVER	192.168.4.1
Database	192.168.4.0	255.255.255.0	192.168.4.4	SYSLOG SERVER	192.168.4.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.1	IDPS ROUTER	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.2	SDN	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.3	FIRE EXIT DOOR	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.4	WINDOW	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.5	FIRE SPRINKLE	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.6	FIRE MONITOR	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.7	FIRE DETECTED LCD SCREEN	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.8	IT SECURITY	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.9	FIRE SIREN	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.10	MOTION SENSOR	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.11	CAMERA	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.12	MAIN ENTRANCE DOOR	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.13	LIGHT	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.14	RESTRICTED AREA DOOR	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.15	ID CARD READER	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.16	TEMPERATURE SENSOR	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.17	TEMPERATURE MONITOR	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.18	HEATER	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.19	AC	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.20	TEMPERATURE LCD SCREEN	192.168.5.1
IT Security and IoT	192.168.5.0	255.255.255.0	192.168.5.100	IoT SERVER	192.168.5.1
Firewall 1	192.168.6.0	255.255.255.0	192.168.6.1	ROUTER2	192.168.6.1
Firewall 1	192.168.6.0	255.255.255.0	192.168.6.2	Firewall 1	192.168.6.1
IDPS Router	192.168.7.0	255.255.255.0	192.168.7.1	IDPS Router	192.168.7.1
IDPS Router	192.168.7.0	255.255.255.0	192.168.7.2	Firewal2 1	192.168.7.1
Firewall 2	192.168.0.0	255.255.255.0	192.168.0.1	ROUTER2	192.168.0.1
Firewall 2	192.168.0.0	255.255.255.0	192.168.0.2	Firewall 2	192.168.0.1
ISP Router	203.0.113.0	255.255.255.0	203.0.113.1	ISP Router	203.0.113.1
ISP Router	203.0.113.0	255.255.255.0	203.0.113.2	Firewall 2	203.0.113.1
Guest	198.51.100.0	255.255.255.0	198.51.100.1	ISP Router	198.51.100.1
Guest	198.51.100.0	255.255.255.0	198.51.100.2	MALICIOUS HOST	198.51.100.1

Table 4.3: IP Address Table

The connection between switches, hosts and IoT devices within the network topology is shown in **Table 4.4**:

Switch	Switch Port	Connection Type	Host/IoT Device	Host Port
Switch 1	Fa0/2	Copper Straight Through	Host1	Fa0
Switch 1	Fa0/3	Copper Straight Through	Host2	Fa0
Switch 2	Fa0/2	Copper Straight Through	Host3	Fa0
Switch 2	Fa0/3	Copper Straight Through	Host4	Fa0
Switch 3	Fa0/2	Copper Straight Through	Reception Desk	Fa0
Switch 3	Fa0/3	Copper Straight Through	Information Desk	Fa0
Switch 4	Fa0/2	Copper Straight Through	DNS Server	Fa0
Switch 4	Fa0/3	Copper Straight Through	Syslog Server	Fa0
Switch 5	Fa0/2	Copper Straight Through	SDN	Fa0
Switch 5	Fa0/3	Copper Straight Through	Fire Exit Door	Fa0
Switch 5	Fa0/4	Copper Straight Through	Window	Fa0
Switch 5	Fa0/5	Copper Straight Through	Fire Sprinkle	Fa0
Switch 5	Fa0/6	Copper Straight Through	Fire Monitor	Fa0
Switch 5	Fa0/7	Copper Straight Through	Fire Detected LCD Screen	Fa0
Switch 5	Fa0/8	Copper Straight Through	IT Security	Fa0
Switch 5	Fa0/9	Copper Straight Through	Fire Siren	Fa0
Switch 5	Fa0/10	Copper Straight Through	Motion Sensor	Fa0
Switch 5	Fa0/11	Copper Straight Through	Camera	Fa0
Switch 5	Fa0/12	Copper Straight Through	Main Entrance Door	Fa0
Switch 5	Fa0/13	Copper Straight Through	Light	Fa0
Switch 5	Fa0/14	Copper Straight Through	Restricted Area Door	Fa0
Switch 5	Fa0/15	Copper Straight Through	ID Card Reader	Fa0
Switch 5	Fa0/16	Copper Straight Through	Temperature Sensor	Fa0
Switch 5	Fa0/17	Copper Straight Through	Temperature Monitor	Fa0
Switch 5	Fa0/18	Copper Straight Through	Heater	Fa0
Switch 5	Fa0/19	Copper Straight Through	AC	Fa0
Switch 5	Fa0/20	Copper Straight Through	IoT Server	Fa0

Table 4.4: The connection between Switches - Host/IoT devices

In the SBMS's network, the routers are configured with the Open Shortest Path First (OSPF) routing protocol to enhance network efficiency and reliability. Applying OSPF is essential for the network as it dynamically controls routing information, assuring that data packets are directed through the most optimal routes. This not only improves the efficiency of network traffic but also ensures the ability to manage a large amount of traffic and maintain uninterrupted operations across different departments. Here is the OSPF routing configuration for the network:

Router	OSPF Networks
Router 1	192.168.1.0
Router 1	192.168.2.0
Router 1	172.16.1.0
Router 1	172.16.3.0
Router 2	192.168.0.0
Router 2	192.168.6.0
Router 2	172.16.1.0
Router 2	172.16.2.0
Router 3	192.168.3.0
Router 3	192.168.4.0
Router 3	172.16.2.0
Router 3	172.16.3.0
IDPS Router	192.168.5.0
IDPS Router	192.168.7.0
IPS Router	203.0.113.0
IPS Router	198.51.100.0

Table 4.5: OSPF Routing Table of Routers

4.2 Router Configuration using CLI

This section focuses on router configuration through the Command Line Interface (CLI), providing an in-depth description of the commands and actions required for setting up the router's configuration for best performance in SBMS's network.

4.2.1 Router 1 Configuration

The basic configuration of ROUTER 1 creates the hostname as "ROUTER1," accompanied by the activation of service password encryption to protect all saved passwords. Cryptographic keys are created using the "crypto key generate rsa" command with a modulus size of 2048. This key is crucial for ensuring the security of SSH communications. The router is additionally protected by an enable secret for privileged access and a username 'SDNnetworksecurity' with a protected password for authorised management.

In **Figure 4.2**, SSH version 2 is enabled for remote management, and the domain name networksecurity.com is configured to simplify the steps of generating cryptographic keys. ROUTER1 is configured with GigabitEthernet ports that have been allocated IP addresses 192.168.1.1 and 192.168.2.1. These ports are set to automatic for the bandwidth and duplex settings.

```
! Set basic configurations
hostname ROUTER1
service password-encryption

! Enable password encryption for security
service password-encryption

! Generate RSA keys for SSH to ensure secure communication
crypto key generate rsa
 modulus 2048

! Set enable secret for security
enable secret 5 $1$mERr$hbENBfUeQECKooMANN4/2/

! Configure username and password for secure access
username SDNnetworksecurity secret 5 $1$mERr$C37ombOqmsN4eydMeSm6r1

! Set SSH and domain configurations
ip ssh version 2
ip domain-name networksecurity.com
ip name-server 192.168.4.2

! Configure interfaces
interface GigabitEthernet0/0/0
 description Connected to Ethernet network
 ip address 192.168.1.1 255.255.255.0
 duplex auto
 speed auto

interface GigabitEthernet0/0/1
 description Connected to Ethernet network
 ip address 192.168.2.1 255.255.255.0
 duplex auto
 speed auto

interface Serial0/1/0
 description Connected to router Router2
 ip address 172.16.1.1 255.255.255.252
 clock rate 56000

interface Serial0/1/1
 description Connected to router Router3
 ip address 172.16.3.2 255.255.255.252

! OSPF configuration
router ospf 1
 router-id 2.1.2.2
 log-adjacency-changes
 auto-cost reference-bandwidth 1000
 network 192.168.1.0 0.0.0.255 area 0
 network 192.168.2.0 0.0.0.255 area 0
 network 172.16.1.0 0.0.0.3 area 0
 network 172.16.3.0 0.0.0.3 area 0

! Set logging servers
logging 192.168.4.2
logging 192.168.4.4

! Configure console and VTY lines
line con 0
 exec-timeout 5 0
 password 7 08220D5D2A16254445
 logging synchronous
 login

line vty 0 4
 exec-timeout 5 0
 password 7 08220D5D2A16254445
 logging synchronous
 login local
 transport input ssh

! NTP Server configuration
ntp server 192.168.4.4
```

Figure 4.2: Router 1 configuration-I

Figure 4.3: Router 1 configuration-II

The Serial0/1/0 interface has been configured with the IP address 172.16.1.1 designed with DCE, provided a clock rate of 56000 and a subnet mask of 255.255.255.252. On the other hand, Serial0/1/1 is set up as DTE with the IP address 172.16.3.2 and the same subnet mask. However, it does not need a clock rate because it receives clocking from another device. ROUTER1 has the OSPF routing protocol enabled, with a router ID of 2.1.2.2. The router actively supports 4 networks within OSPF Area 0, 192.168.1.0, 192.168.2.0, 172.16.1.0, and 172.16.3.0.

The system logging is set up to send data to the IP address 192.168.4.4, which helps with network diagnostics and monitoring. Passwords protect the router's console and VTY lines and are set to automatically log out after five minutes of inactivity. SSH is used for all remote access, providing strong security and operational integrity throughout the network. The Network Time Protocol (NTP) server is set up to synchronise time with the IP address 192.168.4.4, ensuring exact timestamping for logs.

4.2.2 Router 2 Configuration

The main configuration of ROUTER 2 substantially is similar to ROUTER1, with the few changes in settings.

The IP address of ROUTER 2, interface GigabitEthernet0/0/0 is assigned the IP address 192.168.0.1, while GigabitEthernet0/0/1 is assigned the IP address 192.168.6.1. Serial0/1/0 is configured as a DCE with the IP address 172.16.2.1 and a clock rate of 56000. On the other hand, Serial0/1/1 is configured as a DTE and uses the IP address 172.16.1.2 without a clock rate. The OSPF routing protocol on ROUTER2 manage 192.168.0.0, 192.168.6.0, 172.16.1.0, and 172.16.2.0 networks.

```
! Basic Configuration
hostname ROUTER2
service password-encryption

! Security settings
enable secret 5 $1$mERr$hbENBfUeQECKooMANN4/2/
username SDNnetworksecurity secret 5 $1$mERr$C37ombOqmsN4eydMeSm6r1

! Generate RSA keys for SSH
crypto key generate rsa
 modulus 2048

! SSH and Domain Settings
ip ssh version 2
ip domain-name networksecurity.com
ip name-server 192.168.4.2

! Interface Configurations
interface GigabitEthernet0/0/0
 description Connected to Ethernet network
 ip address 192.168.0.1 255.255.255.0
 duplex auto
 speed auto

interface GigabitEthernet0/0/1
 description Connected to Ethernet network
 ip address 192.168.6.1 255.255.255.0
 duplex auto
 speed auto

interface Serial0/1/0
 description Connected to router Router3
 ip address 172.16.2.1 255.255.255.252
 clock rate 56000
```

Figure 4.4: Router 2 configuration-I

```
interface Serial0/1/1
 description Connected to router Router1
 ip address 172.16.1.2 255.255.255.252

! OSPF Configuration
router ospf 1
 router-id 2.1.2.3
 log-adjacency-changes
 auto-cost reference-bandwidth 1000
 network 192.168.0.0 0.0.0.255 area 0
 network 192.168.6.0 0.0.0.255 area 0
 network 172.16.1.0 0.0.0.3 area 0
 network 172.16.2.0 0.0.0.3 area 0

! NTP Server
ntp server 192.168.4.4

! Logging Configuration
logging 192.168.4.4

! Console and VTY Line Security
line con 0
 exec-timeout 5 0
 password 7 08220D5D2A16254445
 logging synchronous
 login

line vty 0 4
 exec-timeout 5 0
 password 7 08220D5D2A16254445
 logging synchronous
 login local
 transport input ssh
```

Figure 4.5: Router 2 configuration-II

4.2.3 Router 3 Configuration

ROUTER3 GigabitEthernet0/0/0 has the IP address 192.168.3.1, and GigabitEthernet0/0/1 has the IP address 192.168.4.1. Serial0/1/0 is set as DCE, assigned the IP address 172.16.3.1, and set the clock rate to 56000. On the other hand, Serial0/1/1 functions as DTE with the IP address 172.16.2.2. The OSPF configuration on ROUTER3 is accurately set up to broadcast the networks 192.168.3.0, 192.168.4.0, 172.16.2.0, and 172.16.3.0.

```
! Basic Configuration
hostname ROUTER3
service password-encryption

! Security settings
enable secret 5 $1$mERr$hbENBfUeQECKooMANN4/2/
username SDNnetworksecurity secret 5 $1$mERr$C37ombOqmsN4eydMeSm6r1

! Generate RSA keys for SSH
crypto key generate rsa
 modulus 2048

! SSH and Domain Settings
ip ssh version 2
ip domain-name networksecurity.com
ip name-server 192.168.4.2

! Interface Configurations
interface GigabitEthernet0/0/0
 description Connected to Ethernet network
 ip address 192.168.3.1 255.255.255.0
 duplex auto
 speed auto

interface GigabitEthernet0/0/1
 description Connected to Ethernet network
 ip address 192.168.4.1 255.255.255.0
 duplex auto
 speed auto

interface Serial0/1/0
 description Connected to router Router1
 ip address 172.16.3.1 255.255.255.252
 clock rate 56000
```

Figure 4.6: Router 3 configuration-I

```
interface Serial0/1/1
 description Connected to router Router2
 ip address 172.16.2.2 255.255.255.252

! OSPF Configuration
router ospf 1
 router-id 2.1.2.4
 log-adjacency-changes
 auto-cost reference-bandwidth 1000
 network 192.168.3.0 0.0.0.255 area 0
 network 192.168.4.0 0.0.0.255 area 0
 network 172.16.2.0 0.0.0.3 area 0
 network 172.16.3.0 0.0.0.3 area 0

! NTP Server
ntp server 192.168.4.4

! Logging Configuration
logging 192.168.4.2
logging 192.168.4.4

! Console and VTY Line Security
line con 0
 exec-timeout 5 0
 password 7 08220D5D2A16254445
 logging synchronous
 login

line vty 0 4
 exec-timeout 5 0
 password 7 08220D5D2A16254445
 logging synchronous
 login local
 transport input ssh
```

Figure 4.7: Router 3 configuration-II

4.2.4 ISP Router Configuration

```
! Basic Configuration
hostname ISP
service password-encryption

! Security settings
enable secret 5 $1$mERr$hbENBfUeQECKooMANN4/2/
username SDNnetworksecurity secret 5 $1$mERr$C37ombOqmsN4eydMeSm6r1

! SSH and Domain Settings
ip ssh version 2
ip domain-name networksecurity.com
ip name-server 192.168.4.2

! Interface Configurations
interface GigabitEthernet0/0/0
 description Connected to external network
 ip address 203.0.113.1 255.255.255.0
 duplex auto
 speed auto

interface GigabitEthernet0/0/1
 description Connected to guest network
 ip address 198.51.100.1 255.255.255.0
 duplex auto
 speed auto

! OSPF Configuration
router ospf 1
 router-id 2.1.2.5
 log-adjacency-changes
 auto-cost reference-bandwidth 1000
 network 198.51.100.0 0.0.0.255 area 0
 network 203.0.113.0 0.0.0.255 area 0

! Logging Configuration
logging 192.168.4.4

! Console and VTY Line Security
line con 0
 exec-timeout 5 0
 password 7 08220D5D2A16254445
 logging synchronous
 login

line vty 0 4
 exec-timeout 5 0
 password 7 08220D5D2A16254445
 logging synchronous
 login local
 transport input ssh

! NTP Server Configuration
ntp server 192.168.4.2
```

Figure 4.8: ISP Router Configuration

The ISP router is set up to handle external connections and provide guest access. The system employs standard security measures such as encrypted passwords and SSH for safe remote administration. Two GigabitEthernet interfaces are configured: one is connected to the Firewall2 with the IP address 203.0.113.1, and the other is connected to the guest network with the IP address 198.51.100.1. OSPF routing facilitates efficient network traffic management between these interfaces by promoting their respective networks.

4.3 Ransomware Payload - Python Script

The main goal of the proposed client and server Python scripts is to simulate a cyber-attack within a controlled environment. In this scenario, a client, malicious host, sends a command to activate ransomware on the server-side IoT device.

4.3.1 Ransomware Payload - Malicious Host Script (Client Python)

The client script is specifically built to initiate a TCP connection using a set server IP address and port number. Once a link is established, it consistently transmits simulated ransomware activation commands every 10 seconds. Every instruction is transmitted as a formatted Protocol Data Unit (PDU) containing a distinct message and command category. The malicious host further receives and handles acknowledgements from the IoT device. In the case of IoT device connection failure, the script will display an error message and terminate any further execution. This simulation demonstrates the malicious host's capacity to execute commands that imitate a situation of a ransomware attack.

```
from tcp import *
from time import sleep
from simulation import *

# Define the IP address and port of the IoT device that Malicious Host will connect to
serverIP = "192.168.0.3"
serverPort = 1025
# Define the protocol and PDU type that will be used for sending data
protocolName = "MyProtocol2"
pduType = "Ransomware"
# Define the color that will be used for the PDU for visualization purposes
pduColor = 0xff00ff

client = TCPClient()    # Create a TCPClient instance for managing the TCP connection

# Callback function that's called when the TCP connection state changes
def onTCPConnectionChange(type):
    # Print the new connection state to the console
    print("Connection state changed to {}".format(type))

# Callback function for handling PDU received from the IoT device
def onTCPReceiveWithPDUInfo(data, pduInfo):
    # Print the response data from the IoT device to the console
    print("Response PDU received with data:", data)

# The main function where the Malicious Host's behavior is defined
def main():
    # Register the callback functions with the Malicious Host instance
    client.onConnectionChange(onTCPConnectionChange)
    client.onReceiveWithPDUInfo(onTCPReceiveWithPDUInfo)

    # Attempt to connect to the server using the provided IP address and port
    if client.connect(serverIP, serverPort):
        # If the connection is successful, print a confirmation message
        print("Connected to server.")
    else:
        # If the connection fails, print an error message and exit the function
        print("Failed to connect to server.")
        return

    # Enter an infinite loop where the Malicious Host periodically sends PDUs to the IoT device
    while True:
        # Create a PDUInfo object with the specified color
        pduInfo = PDUInfo(pduColor)
        # Add a message to the PDU
        pduInfo.addOutMessage("Ransomware trigger command")

        # Set the format of the outgoing PDU with the command to trigger ransomware
        pduInfo.setOutFormat(protocolName, pduType, {
            "type": "COMMAND",
            "command": "TRIGGER_RANSOMWARE"
        })
        # Send the PDU to the IoT device
        client.sendWithPDUInfo("TRIGGER_RANSOMWARE", pduInfo)
        # Print a message to the console confirming that the ransomware trigger was sent
        print("Sent ransomware trigger command.")
        # Pause the loop for 10 seconds before sending the next PDU
        sleep(10)

# Check if the script is run as the main module and, if so, call the main function
if __name__ == "__main__":
    main()
```

Figure 4.9: Ransomware Payload - Malicious Host Python Script

4.3.2 Ransomware Payload - IoT device Script (Server Python)

The server script provides a TCP server for an IoT device, which is waiting for connections on a specified port. When a malicious host establishes a connection, it verifies its acceptance of a ransomware trigger command. When it is identified, it imitates a ransomware attack by encrypting the IoT device's important file, notifying the malicious host of data encryption, and giving details for ransom money. The IoT device confirms its acceptance of commands by transmitting a formatted PDU and keeps an active connection with the host. The process continues indefinitely, simulating the IoT device's constantly ready-to-handle incoming ransomware attacks.

```python
# Import the required modules for handling time delays, TCP communication, and simulation-specific tasks
from time import sleep
from tcp import *
from simulation import *

# Define the port on which the IoT device will listen for incoming connections
port = 1025
# Define the protocol and PDU type used for communication with Malicious Host
protocolName = "MyProtocol2"
pduType = "Ransomware"
# Define the color used for the PDU for visualization
pduColor = 0xff00ff

# Instantiate a TCPServer object to manage IoT device operations
server = TCPServer()

# Define the function that will be called when Malicious Hostt connects to the server
def onTCPNewClient(client):
    # Define a nested function for handling changes in the TCP connection state
    def onTCPConnectionChange(type):
        # When the connection state changes, print a message with the Malicious Host's remote IP and the new state
        print("Connection to " + client.remoteIP() + " changed to state " + str(type))

    # Define a nested function to handle data received from the Malicious Host, along with PDU information
    def onTCPReceiveWithPDUInfo(data, pduInfo):
        # Print a message displaying the Malicious Host's IP and the data received
        print("Received from {} with data: {}".format(client.remoteIP(), data))
        # Check if the received data is a "trigger_ransomware" command
        if data.strip().lower() == "trigger_ransomware":
            # If the command is received, print a simulated ransomware message
            print("""
            WARNING: Your server has been compromised.
            All your data has been encrypted.

            To restore your system, you must pay 0.1 Bitcoin to the following address:
            1BoatSLRHtKNngkdXEeobR76b53LETtpyT

            Instructions on how to acquire and transfer Bitcoin have been sent to your administrator's email address.
            Attempting to remove the software or shutting down the system will result in permanent data loss.
            Your compliance is necessary to ensure the recovery of your data.
```

Figure 4.10: Ransomware Payload - IoT device Python Script I

```python
            """)
            # Add messages to the PDU information acknowledging reception and processing of the command
            pduInfo.addInMessage("Received your command.")
            pduInfo.addOutMessage("Processed your command.")
            # Set the PDU format for the reply to indicate that the command was processed
            pduInfo.setOutFormat(protocolName, pduType, {"type": "REPLY", "data": "Command processed"})
            # Send a response back to the Malicious Host with the updated PDU information
            client.sendWithPDUInfo("Command processed", pduInfo)

    # Register the above nested functions as callbacks to handle events for each client
    client.onConnectionChange(onTCPConnectionChange)
    client.onReceiveWithPDUInfo(onTCPReceiveWithPDUInfo)

# Define a setup function to initialize the server and configure PDUs
def setup():
    # Add a custom PDU format to the simulation with specific fields and properties
    Simulation.addCustomPDU(protocolName, pduType, {
        "title": "My PDU2",
        "units": "Bits",
        "unit_marks": [16],
        "width": 32,
        "fields": [
            {"value": "TYPE: {type}", "size": 32},
            {"value": "DATA: {data}", "size": 32}
        ]
    })
    # Print a message indicating that the server is starting and on which port it's listening
    print("Starting server on port", port)
    # Register the function to handle Malicious Host connections
    server.onNewClient(onTCPNewClient)
    # Start listening for incoming connections on the specified port
    server.listen(port)

# Call the setup function to start the IoT Device server
if __name__ == "__main__":
    setup()
    # Enter an infinite loop to keep the IoT Device server running, checking for new connections or data every 10 seconds
    while True:
        sleep(10)
```

Figure 4.11: Ransomware Payload - IoT device Script II

4.4 Security Components Configuration

This section will present CLI commands for configuring firewalls, SDN, and IDPS to implement security policies, manage network traffic logically, and monitor for potential threats within the network. The setup will demonstrate how each device has been adjusted to meet the network's security needs, ensuring the implementation of strong defence measures and the maintenance of operational performances.

4.4.1 Firewall 1 Configuration

Firewall 1 has been designed to offer strong security and effective traffic control for the network. This configuration is intended to manage access between the internal and external network interfaces, providing a secure setting for the network's operations.

```
! Basic Configuration
hostname FIREWALL1

! Interface Configurations
interface GigabitEthernet1/1
 nameif INSIDE
 security-level 100
 ip address 192.168.6.2 255.255.255.0

interface GigabitEthernet1/2
 nameif OUTSIDE
 security-level 0
 ip address 203.0.113.2 255.255.255.0

! Routing Configuration
route OUTSIDE 0.0.0.0 0.0.0.0 203.0.113.1 1
route INSIDE 192.168.0.0 255.255.255.0 192.168.6.1 1
route INSIDE 192.168.1.0 255.255.255.0 192.168.6.1 1
route INSIDE 192.168.2.0 255.255.255.0 192.168.6.1 1
route INSIDE 192.168.3.0 255.255.255.0 192.168.6.1 1
route INSIDE 192.168.4.0 255.255.255.0 192.168.6.1 1
route INSIDE 192.168.6.0 255.255.255.0 192.168.6.1 1
route INSIDE 172.16.1.0 255.255.255.0 192.168.6.1 1
route INSIDE 172.16.2.0 255.255.255.0 192.168.6.1 1
route INSIDE 172.16.3.0 255.255.255.0 192.168.6.1 1
route INSIDE 192.168.5.0 255.255.255.0 192.168.6.1 1

! Access Control Lists
access-list outside_access_in extended permit icmp any any echo-reply
access-list inside_access_out extended permit icmp any any
access-list outside_access_in extended deny tcp any any eq 1025


! Access Groups
access-group outside_access_in in interface OUTSIDE
access-group inside_access_out out interface INSIDE

! Inspection Policy
class-map inspection_default
 match default-inspection-traffic
policy-map type inspect dns preset_dns_map
 parameters
  message-length maximum 512
policy-map global_policy
 class inspection_default
  inspect dns preset_dns_map
  inspect ftp
  inspect icmp
  inspect tftp
service-policy global_policy global

! OSPF Configuration
router ospf 1
 router-id 2.1.2.1
 log-adjacency-changes
 network 192.168.6.0 255.255.255.0 area 0
 network 203.0.113.0 255.255.255.0 area 0
```

Figure 4.12: Firewall 1 Configuration

Firewall 1 is strategically configured with two main interfaces to effectively manage the flow of traffic. The INSIDE Interface, GigabitEthernet1/1 has been configured with the IP address 192.168.6.2 and a security level of 100, exclusively for linking internal network segments. This configuration implements strict security protocols, successfully protecting confidential internal assets. In contrast, the OUTSIDE Interface, GigabitEthernet1/2, configured with IP address 203.0.113.2 and a security level of 0, controls all incoming traffic from external sources. It serves as the main access point for incoming and outgoing network traffic, assuring that external connections are strictly monitored and regulated.

The OSPF protocol on Firewall 1 uses a router ID of 2.1.2.1 and includes connected networks 192.168.6.0 and 203.0.113.0. Routing is crucial in directing the flow of network data between the internal and external interfaces of Firewall 1. Static routes are implemented to successfully handle this situation and provide dependable and consistent network performance. The gateway at 203.0.113.1 is responsible for handling all incoming external traffic, while internal routes for different network segments, such as 192.168.6.1, are strictly monitored.

Firewall 1 implements Access Control Lists (ACLs) to precisely control the types of traffic permitted in the network. One important ACL that is implemented is called outside_access_in. This ACL particularly allows Internet Control Message Protocol (ICMP) echo-reply messages. The specific setup is essential in the simulation environment of this network. Instead of using HTTP or other commonly used protocols, only ICMP and TCP traffic on port 1025 is utilised to validate the firewall's parameters. The firewall inspection policy, specified in the global_policy, uses class-based policies to thoroughly examine traffic for DNS, FTP, ICMP, and TFTP protocols.

4.4.2 Firewall 2 Configuration

Firewall 2 is configured with IP addresses of OtherDepartment Interface (GigabitEthernet1/1) with 192.168.0.2 to handle external departmental traffic. On the other hand, the INSIDE Interface (GigabitEthernet1/2) is assigned the IP address 192.168.7.2, with a focus on ensuring strong security for internal communications.

Firewall 2 uses static routes to effectively control the flow of data, sending internal network traffic to 198.168.7.1 and managing other department traffic through 192.168.0.1. The network's OSPF configuration improves performance by automatically updating routes for the 192.168.0.0 and 192.168.7.0 networks.

ACLs enhance security by allowing ICMP echo and echo-reply messages for necessary diagnostics while also prohibiting TCP traffic on port 1025. FIREWALL2 incorporates a thorough inspection policy that encompasses DNS, FTP, ICMP, and TFTP.

```
! Basic Configuration
hostname FIREWALL2

! Interface Configurations
interface GigabitEthernet1/1
 nameif OtherDepartment
 security-level 70
 ip address 192.168.0.2 255.255.255.0

interface GigabitEthernet1/2
 nameif INSIDE
 security-level 100
 ip address 192.168.7.2 255.255.255.0

! Routing Configuration
route INSIDE 192.168.7.0 255.255.255.0 198.168.7.1
route INSIDE 192.168.5.0 255.255.255.0 198.168.7.1
route OtherDepartment 192.168.0.0 255.255.255.0 192.168.0.1
route OtherDepartment 192.168.1.0 255.255.255.0 192.168.0.1
route OtherDepartment 192.168.2.0 255.255.255.0 192.168.0.1
route OtherDepartment 192.168.3.0 255.255.255.0 192.168.0.1
route OtherDepartment 192.168.4.0 255.255.255.0 192.168.0.1
route OtherDepartment 192.168.6.0 255.255.255.0 192.168.0.1

! OSPF Configuration
router ospf 1
 router-id 2.1.2.6
 network 192.168.0.0 255.255.255.0 area 0
 network 192.168.7.0 255.255.255.0 area 0

! Access Control Lists
access-list OUTBOUND extended permit icmp any any echo
access-list OUTBOUND extended permit icmp any any echo-reply
access-list OUTBOUND extended deny tcp any any eq 1025

! Apply ACL to interface
access-group OUTBOUND out interface INSIDE

! Inspection Policies
policy-map global_policy
 class inspection_default
  inspect dns preset_dns_map
  inspect ftp
  inspect icmp
  inspect tftp

! Apply the inspection policy globally
service-policy global_policy global
```

Figure 4.13: Firewall 2 Configuration

4.4.3 Intrusion Detection and Prevention System (IDPS) Configuration

At first, the security technology package 'securityk9' was installed using the command 'license boot module c1900 technology-package securityk9', which helps the IDPS router to build an extended range of security features. The hostname is initially configured as IDPS-ROUTER. The GigabitEthernet0/0/0 IP address is 192.168.7.1, and the GigabitEthernet0/0/1 IP address is 192.168.5.1. The router is equipped with an IPS policy called iosips, which enhances security by examining and responding to traffic patterns that correspond to recognised security threats.

The IDPS is set up to save its files in a specific folder on the router's flash memory. It also keeps records of intrusion detection and prevention messages for monitoring purposes. The IPS policy is implemented on the external-facing interface, instructing it to handle the signature with ID 2004 by producing alerts and denying packets inline in case of detecting a threat.

The OSPF routing uses networks 192.168.7.0 and 192.168.5.0 to update the route in the topology. The NTP services are set up to synchronise clocks throughout the network. This configuration demonstrates an active approach to network security, where the IDPS router constantly monitors and regulates all traffic to comply with the SBMS's safety policies.

```
! Basic Configuration
hostname IDPS-ROUTER

! Security and Management
service password-encryption
username SDNnetworksecurity secret 5 $1$mERr$C37ombOqmsN4eydMeSm6r1
ip domain-name networksecurity.com
spanning-tree mode pvst

! Interface Configurations
interface GigabitEthernet0/0/0
 ip address 192.168.7.1 255.255.255.0
 duplex auto
 speed auto

interface GigabitEthernet0/0/1
 ip address 192.168.5.1 255.255.255.0
 ip ips iosips out
 duplex auto
 speed auto

! IPDS Signatures
ip ips config location flash:ipsdirec
ip ips name iosips
ip ips notify log
ip ips signature-category
 category all
  retired true
 category ios_ips basic
  retired false

! IDPS Configuration
int g0/0/1
 ip ips iosips out
```

Figure 4.14: IDPS Configuration - I

```
ip ips signature-definition
 signature 2004 0
  status
   retired false
   enable true
  engine
   event-action produce-alert
   event-action deny-packet-inline

! OSPF Configuration
router ospf 1
 router-id 2.1.2.7
 log-adjacency-changes
 network 192.168.7.0 0.0.0.255 area 0
 network 192.168.5.0 0.0.0.255 area 0

! Line Configurations
line con 0
 exec-timeout 5 0
 password 7 08220D5D2A16254445
 logging synchronous
 login

line vty 0 4
 exec-timeout 5 0
 password 7 08220D5D2A16254445
 logging synchronous
 login
 transport input ssh

! NTP Configuration
ntp server 192.168.5.100
service timestamps log datetime msec
logging host 192.168.5.100
```

Figure 4.15: IDPS Configuration - II

4.4.4 Software-Define Networking (SDN) Configuration

Creating CLI credentials is the first step in establishing access to an SDN controller. These credentials give a unique identity to the user 'SDNnetworksecurity' for secure command-line interactions within the network security domain. The next step, "New Discovery", in setup for SDN configuration, entails the start of a network discovery process applying the Cisco Discovery Protocol (CDP), with the specific objective of targeting the IP address. After network discovery, the SDN system does a thorough network scan. This scan successfully identifies and categorises all devices in the network topology.

Obtaining Service Ticket Through Python Script

The ServiceTicket.py script is a Python programme that automates the process of authenticating a user with a SDN controller and obtaining a service ticket. The service ticket is crucial for all future API calls in the SDN environment, ensuring secure and verified activities.

The script begins by importing the required modules: requests for making HTTP requests and json for handling JSON data. The primary operation, get_auth_ticket, applies a fundamental URL for the SDN API and a login and password to formulate an HTTPS POST request. The credentials are converted into JSON format and transmitted as the body of the request, accompanied by headers that specify the content type.

When the script is executed, it sends a request to the network controller's API. A successful authentication appears if the HTTP response status code is either 200 or 201. The script subsequently deserialises the JSON response to verify the presence of the serviceTicket key, which has the authentic ticket required to access the SDN API. If the script is accessible, it will output the service ticket, confirm the successful retrieval, and provide the necessary response details for debugging purposes.

```python
import requests
import json

def get_auth_ticket(base_uri, username, password):
    # This function sends a POST request to the specified base URI to authenticate a user
    # and retrieve an authentication ticket from the network controller's API.

    headers = {"Content-Type": "application/json"}  # Set the content type of the request to JSON.
    data = json.dumps({"username": username, "password": password})  # Serialize the credentials into JSON.

    # Send the POST request with the serialized JSON credentials.
    response = requests.post("{}/ticket".format(base_uri), data=data, headers=headers)

    # Check if the response status code indicates success (either 200 OK or 201 Created).
    if response.status_code in [200, 201]:
        result = response.json()  # Deserialize the JSON response to a Python dictionary.

        # Check if the expected 'serviceTicket' is in the response dictionary.
        if 'response' in result and 'serviceTicket' in result['response']:
            ticket = result["response"]["serviceTicket"]  # Extract the service ticket.
            # Print the status code, the full response, and the service ticket.
            print("Authentication HTTP Request Status Code: {}".format(response.status_code))
            print("Authentication Response Payload:: {}".format(json.dumps(result, indent=4)))
            print("Service Ticket Obtained:: {}".format(ticket))
            return ticket  # Return the service ticket.
        else:
            # If the ticket isn't in the response, inform the user.
            print("No service ticket found in the response.")
    else:
        # If the response is not successful, print the error status and response text.
        print("Failed to fetch ticket, HTTP Status Code: {}".format(response.status_code))
        print("Response Text: {}".format(response.text))

# This section calls the function using the provided URI, username, and password.
# The obtained ticket, if any, will be stored in the variable 'ticket'.
base_uri = 'http://192.168.5.2/api/v1'
username = "SDNnetworksecurity"
password = "ADMIN123!SDN"
ticket = get_auth_ticket(base_uri, username, password)
```

Figure 4.16: SDN Service Ticket Python Script

By updating the base URI "http://localhost:58000/api/v1" to the controller's real-world address, the script can authenticate and retrieve a service ticket from an actual SDN controller, in **Figure 4.16**

Obtaining Network Device Information using Service Ticket by Pyhton Script

The goal of the NetworkDevices.py script is to communicate with an SDN controller's API in order to verify a user's identity and obtain comprehensive data regarding the network devices within the system. The process begins by the script verifying its identity with the controller's API endpoint using an assigned set of credentials.

```python
import requests
import json

def authenticate(base_uri, username, password):
    """
    Send credentials to the API endpoint to authenticate and get a service ticket.
    The ticket is used in subsequent requests to authenticate the user.
    """
    headers = {"Content-Type": "application/json"}
    payload = json.dumps({"username": username, "password": password})
    response = requests.post("{}/ticket".format(base_uri), data=payload, headers=headers)
    return response.json()["response"]["serviceTicket"]  # Extract the service ticket from the response

def get_network_devices(base_uri, ticket):
    """
    Retrieve a list of network devices from the API using the authenticated service ticket.
    This function makes a GET request to the network-device API endpoint.
    """
    headers = {"X-Auth-Token": ticket}  # Use the service ticket in the request header
    response = requests.get("{}/network-device".format(base_uri), headers=headers)
    return response.json()  # Return the JSON response containing network devices

def print_devices_info(devices):
    """
    Print information about each device in the list.
    The information includes the device hostname, serial number, and software version.
    """
    for device in devices["response"]:  # Loop through each device in the response
        # Print details of each device with labels for clarity
        print("HOSTNAME - {}, SERIAL NUMBER - {}, SOFTWARE VERSION - {}".format(
            device["hostname"], device["serialNumber"], device["softwareVersion"]
        ))

# Set the base URI of the API and credentials for authentication
base_uri = 'http://192.168.5.2/api/v1'
username = "SDNnetworksecurity"
password = "ADMIN123!SDN"

service_ticket = authenticate(base_uri, username, password)  # Authenticate with the API and retrieve a service ticket

network_devices = get_network_devices(base_uri, service_ticket)    # Use the service ticket to get information on network devices

# Print a heading to indicate the upcoming details are about network devices
print("----------------------------------------------------------------")
print("This information about the network devices fetched from the API:")
print("----------------------------------------------------------------")

# Convert the network devices data to a JSON formatted string with an indentation of 4 spaces for readability,
# and print it to the console. This gives a clear view of the network devices' data structure and content.
print(json.dumps(network_devices, indent=4))

print("----------------------------------------------------------------")

# Printing a header for the device information section
print("Device Information:")
print("----------------------------------------------------------------")

# Call the function to print out device details such as hostname, serial number, and software version
print_devices_info(network_devices)
```

Figure 4.17: Obtaining Network Devices using Service Ticket - Python Script

After successfully verifying the user's identity, a service ticket is acquired. This ticket then acts as an access token for further API queries. The script initiates a GET request to the network-device API endpoint, including the service ticket in the request headers to authenticate the connection. The response provides a detailed list of network devices, including their hostnames, serial numbers, and software versions, presented in JSON format.

Obtaining Host Information using Service Ticket by Pyhton Script

The provided Python script functions as an automated tool that interacts with a SDN controller via its API. Once the service ticket is acquired, the script proceeds to gather information regarding the hosts available on the network. The code executes the 'get_hosts' function, which initiates an HTTP GET request to the '/host' endpoint. This function includes the service ticket in the request headers to make sure that the request is authenticated. The API responses include a JSON object that provides information about all the network hosts.

The 'print_devices_info' function receives a list of hosts and iterates through it, extracting and displaying each hostname, IP address, and MAC address.

```python
import requests
import json

def authenticate(base_uri, username, password):
    """
    Send credentials to the API endpoint to authenticate and get a service ticket.
    The ticket is used in subsequent requests to authenticate the user.
    """
    headers = {"Content-Type": "application/json"}
    payload = json.dumps({"username": username, "password": password})
    response = requests.post("{}/ticket".format(base_uri), data=payload, headers=headers)
    return response.json()["response"]["serviceTicket"]  # Extract the service ticket from the response

def get_hosts(base_uri, ticket):
    """
    Retrieve a list of hosts from the API using the authenticated service ticket.
    This function makes a GET request to the network-device API endpoint.
    """
    headers = {"X-Auth-Token": ticket}  # Use the service ticket in the request header
    response = requests.get("{}/host".format(base_uri), headers=headers)
    return response.json()  # Return the JSON response containing network devices

def print_devices_info(devices):
    """
    Print information about each device in the list.
    The information includes the device hostname, serial number, and software version.
    """
    for device in devices["response"]:  # Loop through each device in the response
        # Print details of each device with labels for clarity
        print("HOSTNAME - {}, IP - {}, MAC - {}".format(
            device["hostName"], device["hostIp"], device["hostMac"]
        ))

# Set the base URI of the API and credentials for authentication
base_uri = 'http://192.168.5.2/api/v1'
username = "SDNnetworksecurity"
password = "ADMIN123!SDN"

# Authenticate with the API and retrieve a service ticket
service_ticket = authenticate(base_uri, username, password)

# Use the service ticket to get information on network devices
host_devices = get_hosts(base_uri, service_ticket)

# Print a heading to indicate the upcoming details are about host devices
print("-----------------------------------------------------------------")
print("This information about the host devices fetched from the API:")
print("-----------------------------------------------------------------")

# Convert the host devices data to a JSON formatted string with an indentation of 4 spaces for readability,
# and print it to the console. This gives a clear view of the host devices' data structure and content.
print(json.dumps(host_devices, indent=4))

# Printing a header for the device information section
print("-----------------------------------------------------------------")
print("Device Information:")
print("-----------------------------------------------------------------")

# Call the function to print out device details such as hostname, serial number, and software version
print_devices_info(host_devices)
```

Figure 4.18: Obtaining Host Information Using Service Ticket - Python Script

4.5 Smart Fire Alert System

This code is to set up a smart fire alert system. It is specifically built to communicate with different IoT components using GPIO pins. The system constantly monitors a fire sensor and, upon detecting a fire, initiates safety protocols such as activating sprinklers, opening doors and windows, updating an LCD display, and sounding a siren. If there is a constant absence of fire detection, the systems are deactivated, and the LCD is updated to indicate that everything is safe. The script operates in a continuous loop, ensuring constant monitoring and prompt reaction to fire occurrences.

```
# Import specific functions from the GPIO module to interact with the hardware pins
from gpio import pinMode, digitalRead, customWrite, INPUT, OUTPUT
# Import the time and sleep function from the time module to add delays in the program
from time import time, sleep

# Define constants for the GPIO pin numbers for better readability and maintainability
fire_pin = 0  # The pin number for the fire sensor input
sprinkler_pin = 1  # The pin to control the sprinkler system
door_pin = 2  # The pin to control the door
window_pin = 3  # The pin to control the window
lcd_pin = 4  # The pin connected to an LCD display for status messages
siren_pin = 5  # The pin to control the siren
fire_detection_value = 1023  # The sensor value indicating fire detection

def fire_alert_system():
    # Set up the GPIO pins with the appropriate input or output designation
    pinMode(fire_pin, INPUT)  # Set the fire sensor pin as input
    pinMode(sprinkler_pin, OUTPUT)  # Set the sprinkler control pin as output
    pinMode(door_pin, OUTPUT)  # Set the door control pin as output
    pinMode(window_pin, OUTPUT)  # Set the window control pin as output
    pinMode(lcd_pin, OUTPUT)  # Set the LCD display pin as output
    pinMode(siren_pin, OUTPUT)  # Set the siren control pin as output

    # Print a starting message to indicate that the fire alert system is now running
    print("FIRE ALERT SYSTEM")
    consecutive_no_fire_count = 0  # Initialize a counter to track the number of consecutive no-fire readings

    # Enter an infinite loop to continuously check the fire sensor
    while True:
        fire_value = digitalRead(fire_pin)  # Read the value from the fire sensor

        # Check if the sensor value corresponds to the defined fire detection value
        if fire_value == fire_detection_value:
            handle_fire_detected()  # Call a function to handle fire detection actions
            consecutive_no_fire_count = 0  # Reset the no-fire count upon fire detection
        else:
            # If no fire is detected, increment the counter and call the no-fire handler
            consecutive_no_fire_count += 1  # Increment the no-fire count
            print("Consecutive no-fire count: {}".format(consecutive_no_fire_count))  # Print the count for logging purposes
            handle_no_fire()  # Call a function to handle the no-fire actions

        # If the system detects no fire for 10 consecutive readings, exit the loop
        if consecutive_no_fire_count >= 10:
            print("No fire detected for 10 consecutive iterations - Exiting loop")
            break  # Break out of the loop to stop the system
```

Figure 4.19: Smart Fire Alert System - Part I

```
        sleep(1)  # Wait for 1 second before the next loop iteration to reduce CPU usage

# Define the function to handle actions when fire is detected
def handle_fire_detected():
    print("FIRE DETECTED")  # Print a message indicating fire detection
    customWrite(sprinkler_pin, '1')  # Activate the sprinkler system
    customWrite(door_pin, '1,0')  # Open the doors
    customWrite(window_pin, '1')  # Open the windows
    customWrite(lcd_pin, 'FIRE DETECTED')  # Update the LCD display with the fire detection message
    customWrite(siren_pin, '1')  # Activate the siren

    # Begin the evacuation announcement procedure when a fire is detected
    last_announcement_time = time() # Store the initial time when the announcement loop starts
    while True:  # Start an infinite loop to keep announcing until the fire is no longer detected
        current_time = time()  # Get the current time during each iteration of the loop

        # Check if 5 seconds have passed since the last announcement
        if current_time - last_announcement_time >= 5:
          # Print the evacuation message to the console, which would be equivalent to broadcasting the announcement in a real system
            print("Attention please: This is an automated safety notification. "
                  "A fire incident has been detected in the building. "
                  "Please evacuate immediately through the nearest fire exit. "
                  "Do not use elevators. Repeat, do not use elevators. "
                  "Proceed calmly to the closest emergency exit and leave the building. "
                  "Thank you for your cooperation.")

            # Update the last announcement time to the current time after the announcement
            last_announcement_time = current_time

        # Check if the fire is still detected
        if digitalRead(fire_pin) != fire_detection_value:
            break # Exit the while loop if no fire is detected, which ends the evacuation announcements
        sleep(0.1)  # A short sleep to prevent this loop from hogging CPU

# Define the function to handle actions when no fire is detected
def handle_no_fire():
    customWrite(sprinkler_pin, '0')  # Deactivate the sprinkler system
    customWrite(door_pin, '0,1')  # Close the doors
    customWrite(window_pin, '0')  # Close the windows
    customWrite(lcd_pin, 'ALL IS WELL')  # Update the LCD display with an all-clear message
    customWrite(siren_pin, '0')  # Deactivate the siren

# Main entry point of the script
if __name__ == '__main__':
    fire_alert_system()  # Call the fire alert system function to start the system
```

Figure 4.20: Smart Fire Alert System - Part II

4.6 Smart Motion Detection System

This script implements a smart motion detection system that uses GPIO pins to regulate different IoT devices such as cameras, lights, and doors. When motion is detected, the system initiates the camera, opens the door, and activates the light, followed by printing a confirmation of the operation. When no movement is detected, the system disables the devices and ensures that the door is locked. It keeps track of the number of consecutive no-motion events to decide when to terminate the loop.

```
from gpio import *
from time import *

# Constants for the GPIO pin numbers and motion detected value
motion_sensor_pin = 6
camera_pin = 7
door_pin = 8
light_pin = 9
motion_detected_value = 1023  # The value indicating motion is detected

def setup_pins():
    """Set up GPIO pins as input or output as required."""
    pinMode(motion_sensor_pin, INPUT)
    pinMode(camera_pin, OUTPUT)
    pinMode(door_pin, OUTPUT)
    pinMode(light_pin, OUTPUT)

def motion_detected_actions():
    """Actions to perform when motion is detected."""
    customWrite(camera_pin, '1')  # Activate Camera
    customWrite(door_pin, '1,0')  # Unlock/Open Door
    customWrite(light_pin, '2')  # Turn on Light
    print("Actions executed: Camera activated, Door opened, Light turned on.")

def no_motion_actions():
    """Actions to perform when no motion is detected."""
    customWrite(camera_pin, '0')  # Deactivate Camera
    customWrite(door_pin, '0,0')  # Lock/Close Door
    customWrite(light_pin, '0')  # Turn off Light
    print("Actions executed: Camera deactivated, Door closed, Light turned off.")

def main():
    print("Starting smart motion detection system")
    setup_pins()

    no_motion_count = 0  # Initialize counter for consecutive no-motion readings

    while True:
        if digitalRead(motion_sensor_pin) == motion_detected_value:
            print("Security Alert: Motion has been detected near the reception hall. "
               "An individual may be approaching. Please verify their identification immediately.")
            no_motion_count = 0  # Reset the no-motion counter
            motion_detected_actions()
        else:
            no_motion_count += 1
            print("No motion detected. Count: {}".format(no_motion_count))
            no_motion_actions()

            # Exit the loop if no motion is detected for a predefined number of times
            if no_motion_count >= 10:
                print("No motion detected for 10 consecutive checks - Exiting system.")
                break

        sleep(1)  # Wait for 1 second before checking again

if __name__ == '__main__':
    main()
```

Figure 4.21: Smart Motion Detection System - Python Script

4.7 Smart Temperature Control System

The code for the Smart Temperature Control System automates the regulation of a room's air conditioning (AC) and heating by utilising temperature data from a sensor. The script configures the GPIO pins to set up communication with the air conditioner, heater, and LCD display. The air conditioning system is triggered when the temperature is beyond 520, while the heater is activated when the temperature falls below 510. Under normal circumstances, both systems are deactivated in order to maintain a comfortable temperature. The LCD screen is updated to display the system's current state, giving a visual representation of its condition. The procedure loops at specific times, consistently observing and modifying the temperature of the area to achieve the optimal level of comfort and efficiency.

```
from gpio import *
# Import the time and sleep function from the time module to add delays in the program
from time import time, sleep

# Constants for the GPIO pin numbers and temperature thresholds
temperature_sensor_pin = 0
ac_pin = 3
heater_pin = 2
lcd_pin = 1
temp_high_threshold = 520  # The temperature value that triggers the AC
temp_low_threshold = 510   # The temperature value that triggers the heater

# Initialize GPIO pins
def setup_pins():
    pinMode(temperature_sensor_pin, INPUT)
    pinMode(ac_pin, OUTPUT)
    pinMode(heater_pin, OUTPUT)
    pinMode(lcd_pin, OUTPUT)

# Activate the air conditioning system and update the LCD
def turn_on_ac():
    digitalWrite(ac_pin, HIGH)
    customWrite(lcd_pin, "AC-ON")
    print("Air conditioning activated.")

# Activate the heating system and update the LCD
def turn_on_heater():
    digitalWrite(heater_pin, HIGH)
    customWrite(lcd_pin, "HEATER-ON")
    print("Heating system activated.")

# Turn off all temperature control systems and update the LCD to indicate normal temperature
def maintain_normal_temp():
    digitalWrite(ac_pin, LOW)
    digitalWrite(heater_pin, LOW)
    customWrite(lcd_pin, "NORMAL-TEMP")
    print("Temperature is normal. All systems standby.")

def main():
    print("Starting SMART ROOM TEMPERATURE SYSTEM")
    setup_pins()

    while True:
        # Read temperature from the sensor
        temp = digitalRead(temperature_sensor_pin)
        print("Temperature Reading: {}".format(temp))

        # Determine and apply temperature control actions
        if temp >= temp_high_threshold:
            turn_on_ac()
        elif temp < temp_low_threshold:
            turn_on_heater()
        else:
            maintain_normal_temp()

        sleep(3)  # Wait for 3 seconds before the next read

if __name__ == "__main__":
    main()
```

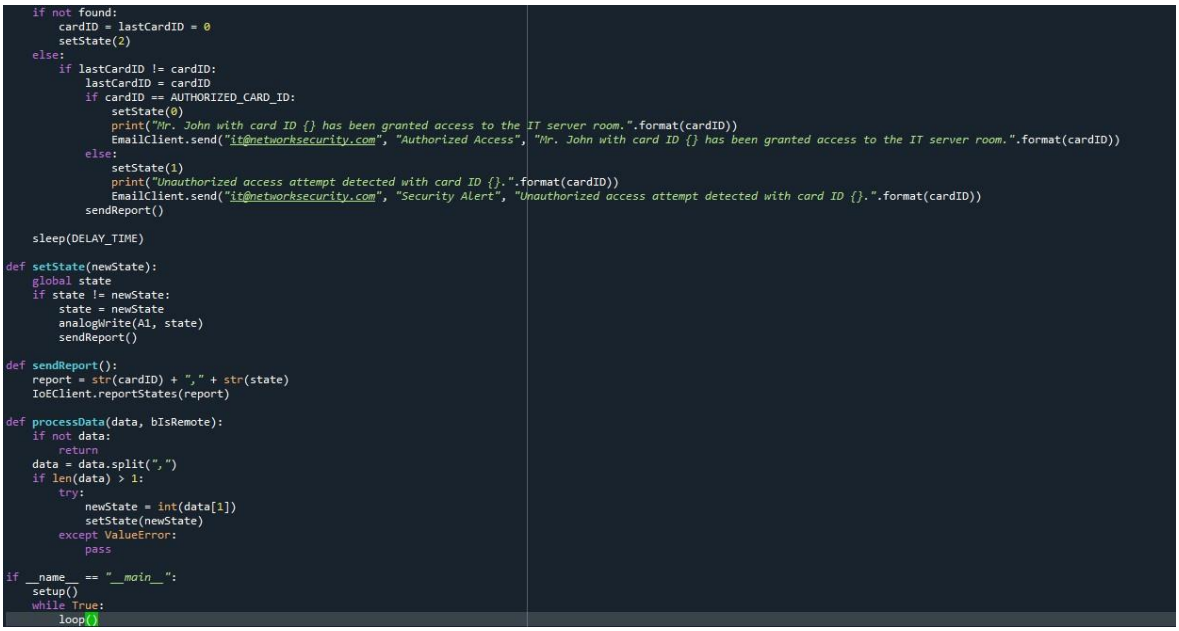
Figure 4.22: Smart Temperature Control System - Python Script

4.8 Smart ID Card Reader System

The Smart ID Card Reader System code is a security system that constantly scans for RFID cards to access the restricted area. Upon detecting a card, the script proceeds to verify the card's ID with an authorised list. If the ID matches the authorised card, access is granted, the system updates to a "Valid" status, and an email notification is sent out, confirming the access of a recognised individual to the restricted area. On the other hand, if a card is not recognised, the status will be changed to "Invalid," and an alert will be sent out, indicating an unauthorised attempt to get access. The system's condition is also transmitted to the Internet of Everything (IoE) server for the purpose of recording and monitoring.

```python
# Import various modules needed for the simulation and control of the RFID reader and email notifications
from options import Options
from time import sleep
from physical import *
from gpio import *
from ioeclient import IoEClient
from email import *  # Assume EmailClient is defined elsewhere as shown earlier

# Set constants for the delay between loops and the read distances for the RFID reader
delay_time = 1  # Delay time in seconds
x_read_distance = 50
y_read_distance = 50

# Initialize variables to store the current and last read card IDs, and the current state of the reader
cardID = 0
lastCardID = 0
state = 2  # waiting

# Define an authorized card ID for access control
AUTHORIZED_CARD_ID = 1001

# The setup function configures the IoE (Internet of Everything) client for the RFID reader
def setup():
    IoEClient.setup({
        "type": "RFID Reader",
        "states": [
            {"name": "Card ID", "type": "number", "unit": '', "controllable": False},
            {"name": "Status", "type": "options", "options": {"0": "Valid", "1": "Invalid", "2": "Waiting"}, "controllable": True}
        ]
    })
    # Set up a callback to process data when it is received by the IoE client
    IoEClient.onInputReceive(lambda rinput: processData(rinput, True))
    # Configure the EmailClient with the credentials for sending emails
    EmailClient.setup("iot@networksecurity.com", "networksecurity.com", "IoT", "iOt123!")

# The loop function is the main execution loop of the RFID reader
def loop():
    global cardID, lastCardID, state  # Declare globals to modify them within the function
    # Scan for devices within the specified range
    devices = devicesAt(getCenterX(), getCenterY(), x_read_distance, y_read_distance)
    found = False  # A flag to check if a valid card is found
    for device in devices:
        if device == getName():  # Skip if the detected device is the reader itself
            continue

        # Try to get the card ID property from the detected device
        tempCardID = getDeviceProperty(device, 'CardID')
        if tempCardID:
            try:
                tempCardID = int(tempCardID)  # Convert to an integer
                cardID = tempCardID  # Set the card ID
                found = True  # Mark that we've found a card
                break
            except ValueError:
                continue  # Ignore the device if the card ID is not an integer
```

Figure 4.23: Smart ID Card Reader System - Python Script



```
    if not found:
        cardID = lastCardID = 0
        setState(2)
    else:
        if lastCardID != cardID:
            lastCardID = cardID
            if cardID == AUTHORIZED_CARD_ID:
                setState(0)
                print("Mr. John with card ID {} has been granted access to the IT server room.".format(cardID))
                EmailClient.send("it@networksecurity.com", "Authorized Access", "Mr. John with card ID {} has been granted access to the IT server room.".format(cardID))
            else:
                setState(1)
                print("Unauthorized access attempt detected with card ID {}.".format(cardID))
                EmailClient.send("it@networksecurity.com", "Security Alert", "Unauthorized access attempt detected with card ID {}.".format(cardID))
            sendReport()

    sleep(DELAY_TIME)

def setState(newState):
    global state
    if state != newState:
        state = newState
        analogWrite(A1, state)
        sendReport()

def sendReport():
    report = str(cardID) + "," + str(state)
    IoEClient.reportStates(report)

def processData(data, bIsRemote):
    if not data:
        return
    data = data.split(",")
    if len(data) > 1:
        try:
            newState = int(data[1])
            setState(newState)
        except ValueError:
            pass

if __name__ == "__main__":
    setup()
    while True:
        loop()
```

Figure 4.24: Smart ID Card Reader System - Python Script

```
# Process data received from the IoE server
def processData(data, bIsRemote):
    if not data:
        return
    # Split the received data into parts and try to set the state
    data = data.split(",")
    if len(data) > 1:
        try:
            newState = int(data[1])
            setState(newState)
        except ValueError:
            pass  # Ignore if the data cannot be converted to an integer

# Execute the setup function and then run the loop indefinitely
if __name__ == "__main__":
    setup()
    while True:
        loop()
```

Figure 4.25: Smart ID Card Reader System - Python Script

”At the end of the day, the goals are simple: safety and security.”

— Jodi Rell

Chapter 5

SIMULATION TESTING

The testing chapter thoroughly assesses the Smart IoT interconnection and network safety features of the Smart Business Management System. It also thoroughly evaluates the durability of each component against ransomware through a range of staged scenarios. The chapter provides an overview of the techniques and outcomes of tests performed to validate the success of security systems, ensuring the integrity and dependability of the system’s infrastructure.

5.1 Scenario 1: Network Connectivity Testing

Scenario 1 is important for this phase, as it includes a series of ping tests that are essential in verifying the network’s operational status and the effectiveness of routing configurations.

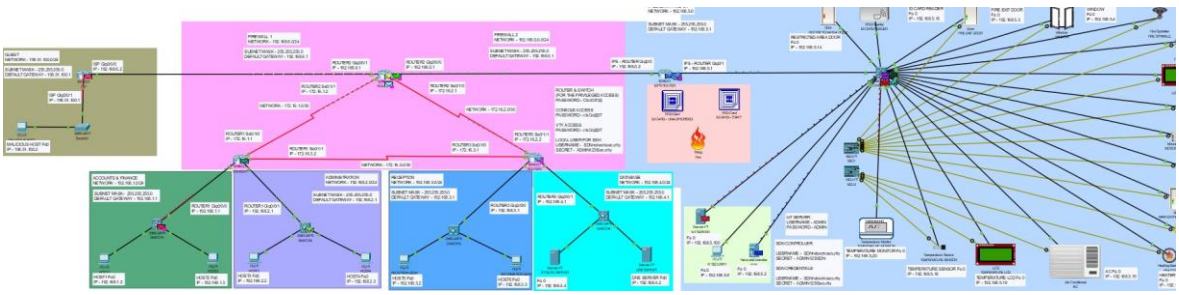
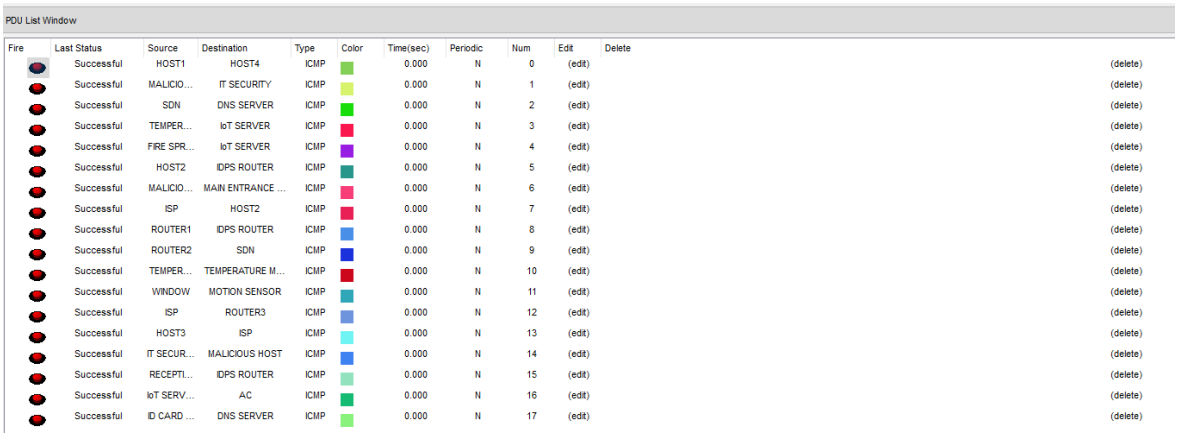


Figure 5.1: Simulating a series of pings throughout the network

The simulation phase provides a detailed overview of the network, highlighting the active use of the ICMP

protocol to evaluate network connectivity. By performing ping tests, every section of the topology is thoroughly tested, showing the smooth integration of network devices and the stability of communication. The visual representation demonstrates the logical and active process of testing a network, confirming the efficiency of operations and the quick exchange of data between the connected nodes.



PDU List Window

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
	Successful	HOST1	HOST4	ICMP		0.000	N	0	(edit)	(delete)
	Successful	MALICIO...	IT SECURITY	ICMP		0.000	N	1	(edit)	(delete)
	Successful	SDN	DNS SERVER	ICMP		0.000	N	2	(edit)	(delete)
	Successful	TEMPER...	IoT SERVER	ICMP		0.000	N	3	(edit)	(delete)
	Successful	FIRE SPR...	IoT SERVER	ICMP		0.000	N	4	(edit)	(delete)
	Successful	HOST2	IDPS ROUTER	ICMP		0.000	N	5	(edit)	(delete)
	Successful	MALICIO...	MAIN ENTRANCE ...	ICMP		0.000	N	6	(edit)	(delete)
	Successful	ISP	HOST2	ICMP		0.000	N	7	(edit)	(delete)
	Successful	ROUTER1	IDPS ROUTER	ICMP		0.000	N	8	(edit)	(delete)
	Successful	ROUTER2	SDN	ICMP		0.000	N	9	(edit)	(delete)
	Successful	TEMPER...	TEMPERATURE M...	ICMP		0.000	N	10	(edit)	(delete)
	Successful	WINDOW	MOTION SENSOR	ICMP		0.000	N	11	(edit)	(delete)
	Successful	ISP	ROUTER3	ICMP		0.000	N	12	(edit)	(delete)
	Successful	HOST3	ISP	ICMP		0.000	N	13	(edit)	(delete)
	Successful	IT SECUR...	MALICIOUS HOST	ICMP		0.000	N	14	(edit)	(delete)
	Successful	RECEPTI...	IDPS ROUTER	ICMP		0.000	N	15	(edit)	(delete)
	Successful	IoT SERV...	AC	ICMP		0.000	N	16	(edit)	(delete)
	Successful	ID CARD ...	DNS SERVER	ICMP		0.000	N	17	(edit)	(delete)

Figure 5.2: Network ping results

The **Figure 5.2** confirmed the successful ping tests throughout the network, suggesting a reliable and effective connection between different devices and nodes. Every ICMP packet, shown by the consistent 'Successful' status, highlights a network that is set up for stable communication.

5.2 Scenario 2: Smart Fire Alert System Testing

The IoT devices shown in **Figure 5.3** build an interconnected system that responds in collaboration. In this scenario, the fire monitor consistently scans for traces of fire. When it is detected, it activates the safety protocol, which manages various devices to ensure a safe and organised evacuation.

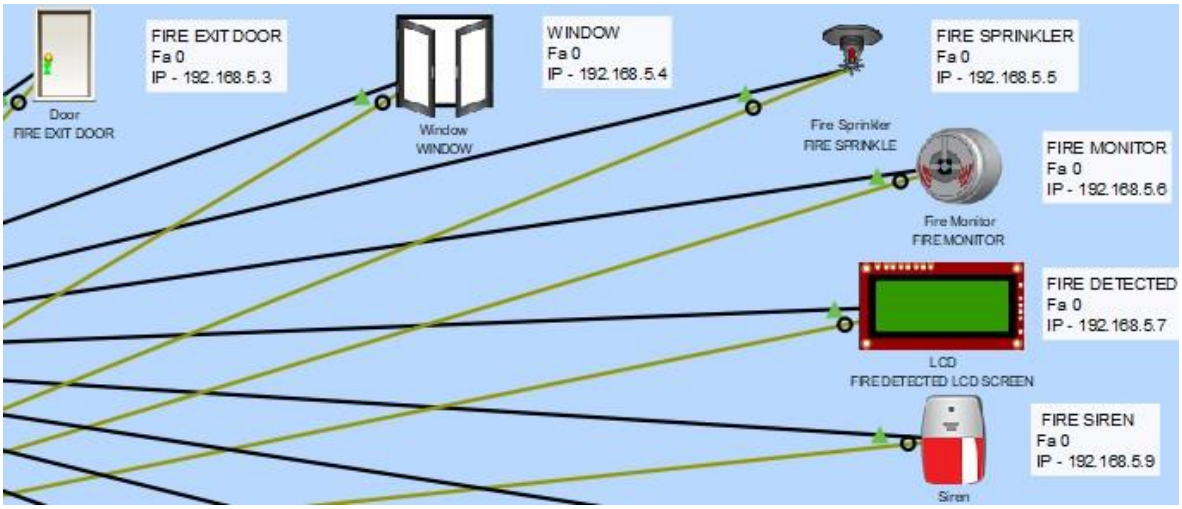


Figure 5.3: Testing Smart Fire Alert System

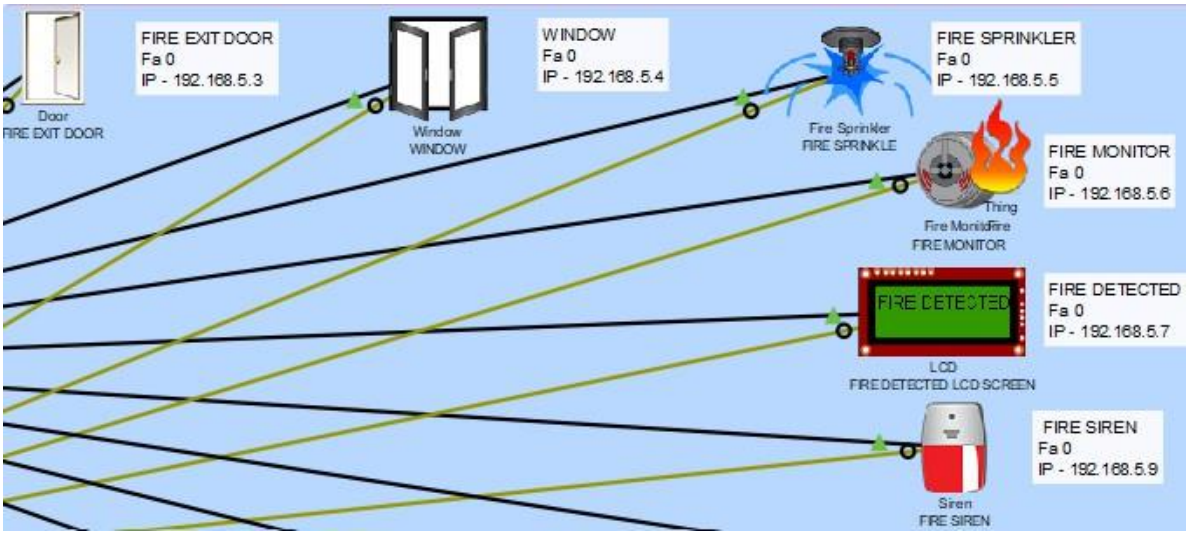


Figure 5.4: Smart Fire Alert System result

The system performed successful execution during the fire scenario test, **Figure 5.4**. Upon detecting the fire, the system carried out a sequence of activities, including opening the fire exit door and windows, activating sprinklers, making a loud sound through the siren and displaying an alert by LCD screen for evacuation safely.

```
Attention please: This is an automated safety notification. A fire incident has been detected in the
building. Please evacuate immediately through the nearest fire exit. Do not use elevators. Repeat, do
not use elevators. Proceed calmly to the closest emergency exit and leave the building. Thank you for
your cooperation.
```

Figure 5.5: Evacuation alert during fire event

The **Figure 5.5** displays an automatic safety announcement activated by the fire alert system. It instructs people to evacuate immediately through the closest fire exit in the event of a detected fire.

```
Consecutive no-fire count: 1
Consecutive no-fire count: 2
Consecutive no-fire count: 3
Consecutive no-fire count: 4
Consecutive no-fire count: 5
Consecutive no-fire count: 6
Consecutive no-fire count: 7
Consecutive no-fire count: 8
Consecutive no-fire count: 9
Consecutive no-fire count: 10
No fire detected for 10 consecutive iterations - Exiting loop
FireAlert (Python) finished running.
```

Figure 5.6: Smart Fire Alert System ends monitoring after confirming no signs of fire

Figure 5.6 demonstrates the result of the Smart Fire Alert System following a successful operation. The system recorded a sequence of ten consecutive seconds where no fire was detected, which caused the protocol to properly terminate the monitoring loop, indicating the end of the fire alert activity.

5.3 Scenario 3: Smart Motion Detection System Testing

Figure 5.7 is a test of the Smart Motion Detection System to ensure it functions accordingly. This test emulates the system's reaction to detect motion, confirming the synchronised operation of a camera, unlocking the main entrance door, and turning on lights.

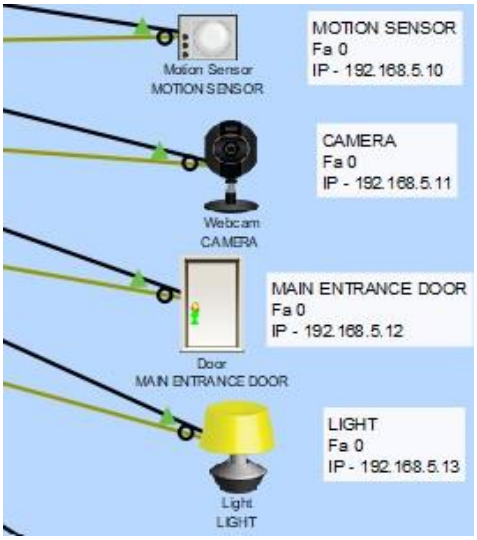


Figure 5.7: Testing Smart Motion Detection System

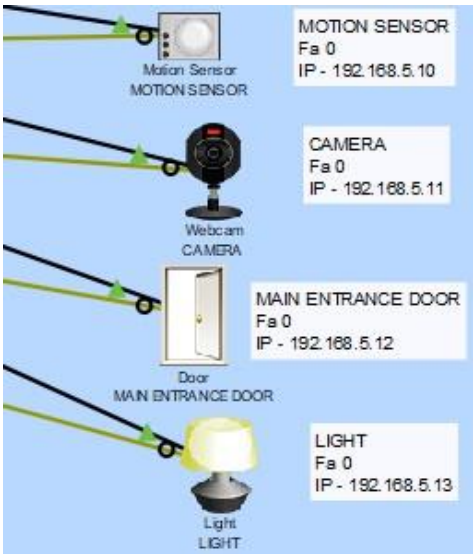


Figure 5.8: Smart Motion Detection System result

During the testing phase **Figure 5.8**, the Smart Motion Detection System was successfully activated in response to motion by using the alt+mouse cursor movement on the sensor to imitate motion detection. When motion was detected, the system activated the security camera, unlocked the main entrance door, and activated the light.

```
Security Alert: Motion has been detected near the reception hall. An individual
may be approaching. Please verify their identification immediately.
Actions executed: Camera activated, Door opened, Light turned on.
No motion detected. Count: 1
Actions executed: Camera deactivated, Door closed, Light turned off.
No motion detected. Count: 2
Actions executed: Camera deactivated, Door closed, Light turned off.
No motion detected. Count: 3
Actions executed: Camera deactivated, Door closed, Light turned off.
No motion detected. Count: 4
Actions executed: Camera deactivated, Door closed, Light turned off.
No motion detected. Count: 5
Actions executed: Camera deactivated, Door closed, Light turned off.
No motion detected. Count: 6
Actions executed: Camera deactivated, Door closed, Light turned off.
No motion detected. Count: 7
Actions executed: Camera deactivated, Door closed, Light turned off.
No motion detected. Count: 8
Actions executed: Camera deactivated, Door closed, Light turned off.
No motion detected. Count: 9
Actions executed: Camera deactivated, Door closed, Light turned off.
No motion detected. Count: 10
Actions executed: Camera deactivated, Door closed, Light turned off.
No motion detected for 10 consecutive checks - Exiting system.
MotionAlert (Python) finished running.
```

Figure 5.9: Smart Motion Detection System ends monitoring after confirming no signs of motion

Figure 5.9 shows the system switched into a monitoring phase, keeping track of the duration during which no motion was detected. After detecting 10 consecutive instances of no motion, the system determined that there was no longer any movement and stopped functioning.

5.4 Scenario 4: Smart Temperature Control System Testing

Figure 5.10 is testing the Smart Temperature Control System, which features a circuit containing a temperature monitor, sensor, LCD display, air conditioning, and heating elements. The setup was designed to continually track and adjust temperature levels within the environment to maintain normal temperature.

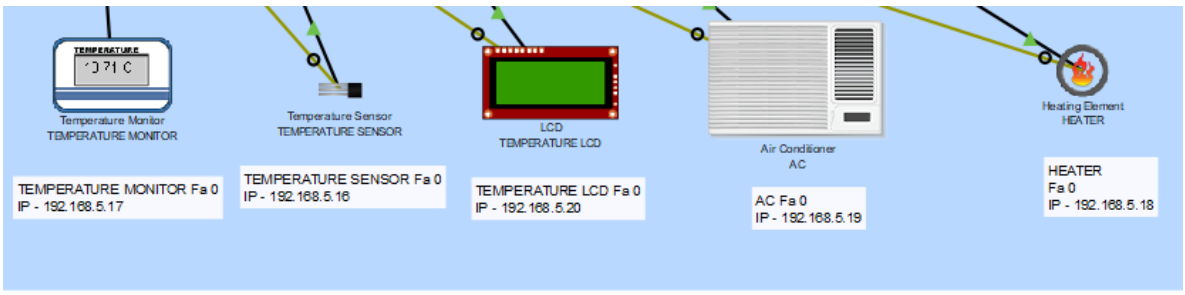


Figure 5.10: Testing Smart Temperature Control System

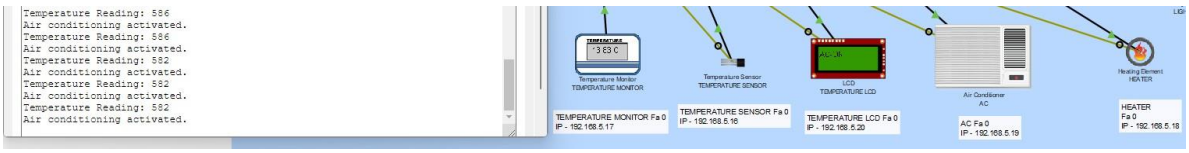


Figure 5.11: Smart Temperature Control System high-temperature result

Figure 5.11 shows the current temperature reading is high, which is above 520, causing the air conditioning to activate, as shown by the blinking red light on the AC unit. The LCD panel shows the activation of the air conditioning system by displaying the message ”AC on,” indicating that the system is now operating to lower

the environment's temperature.

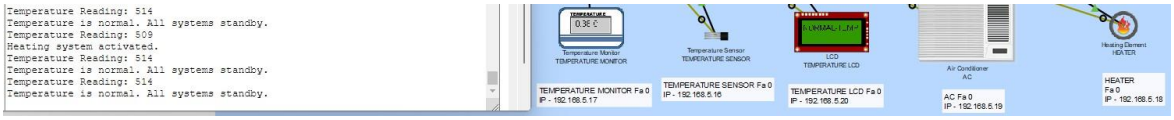


Figure 5.12: Smart Temperature Control System normal temperature result

Figure 5.12 indicates that the temperature readings have reached an ideal state inside the normal range, causing the smart system to activate standby mode for all systems. The LCD panel shows a status update indicating that 'Normal Temperature' has been reached and the AC system is no longer running.

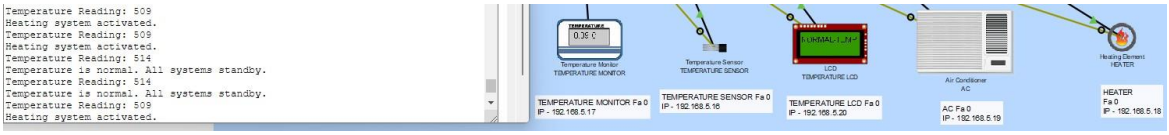


Figure 5.13: Smart Temperature Control System low-temperature result

When the temperature dropped in **Figure 5.13**, the smart system responded by activating the heating system. The temperature readings, changing within the set limits, activated the heater to maintain a warm environment. The system's adaptability is demonstrated by its quick activation and deactivation, ensuring comfort without the need for individual interaction.

5.5 Scenario 5: Smart ID Card Reader System Testing

Figure 5.14 involves verifying the Smart ID Card Reader System, which aims to enhance security by controlling entry to restricted areas. The system's success is measured by its capacity to verify credentials and control access rights in real-time.

Figure 5.14: Testing Smart ID Card System

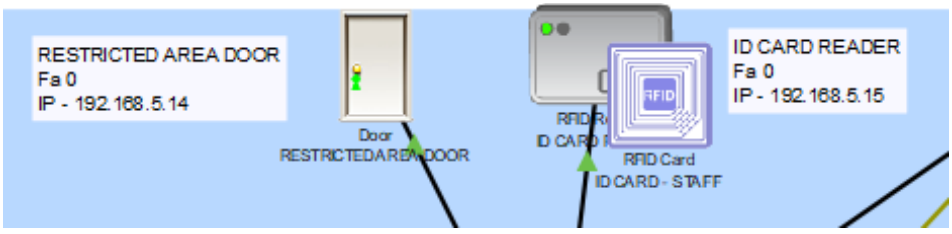


Figure 5.15: Smart ID Card System - Authorised ID card

Figure 5.15 displays the Smart ID Card Reader System granting access authorisation. The reader verifies the authenticity of an ID card, causing a green light to blink and allowing entrance to the individual, which shows the successful operation of the access control system.

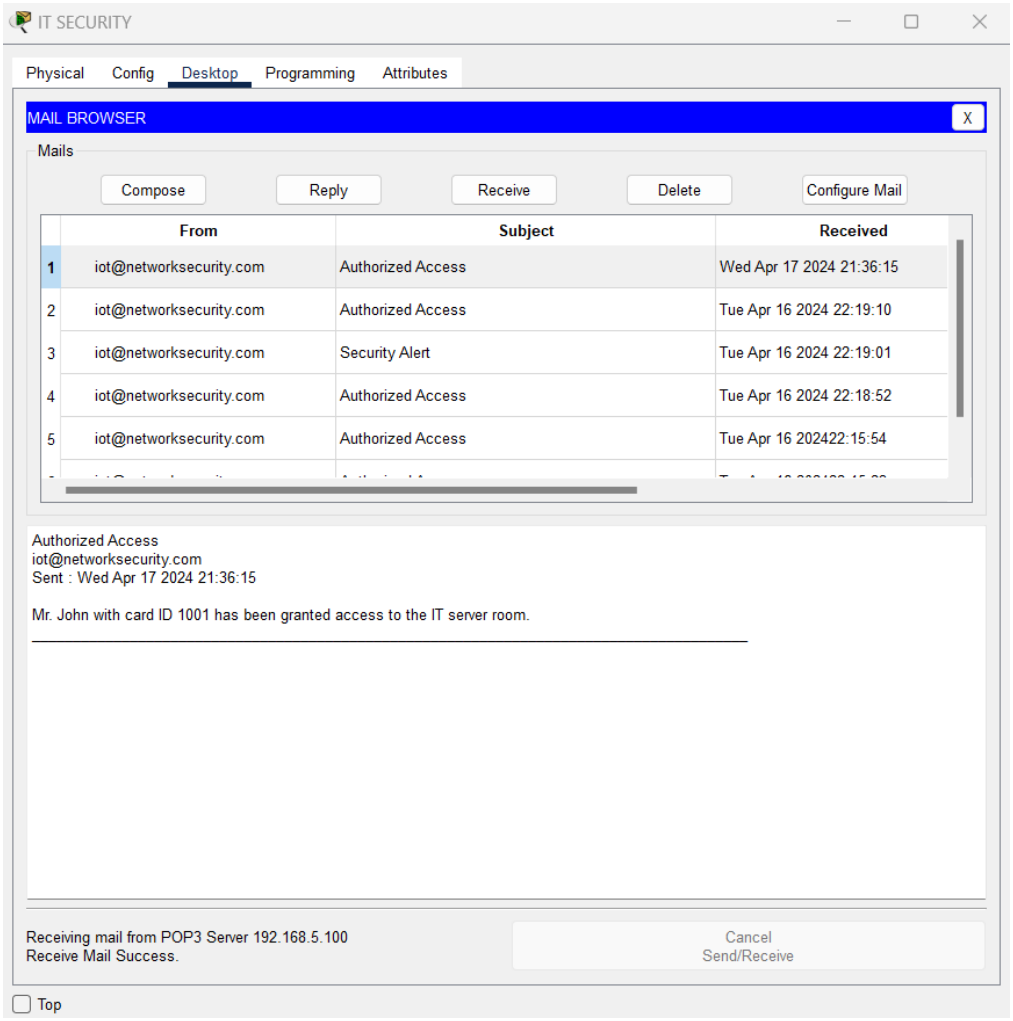


Figure 5.16: Smart ID Card System email alert - I

The ”IT Security” email in **Figure 5.16** shows the Smart ID Card Reader’s notification system. The IoT Server sent the email from the ID Card Reader with a heading that includes ”Authorised Access.” The email verifies that an individual named ”Mr. John” with card identification number 1001 has been authorised to enter the restricted area, demonstrating that the system is properly managing and documenting access control actions.

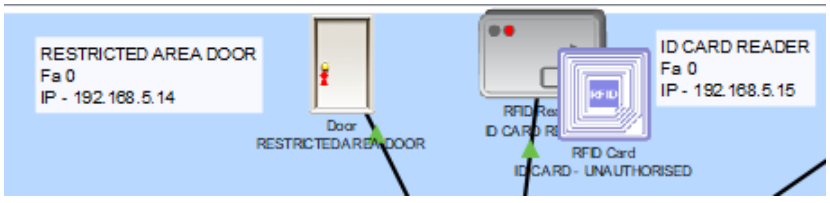


Figure 5.17: Smart ID Card System - Unauthorised ID card

Figure 5.17 and **Figure 5.18** represents the Smart ID Card Reader System denying access to the restricted area because of an invalid ID card. The red light on the scanner signifies that the card is invalid, which means that the door stays closed and prevents access.

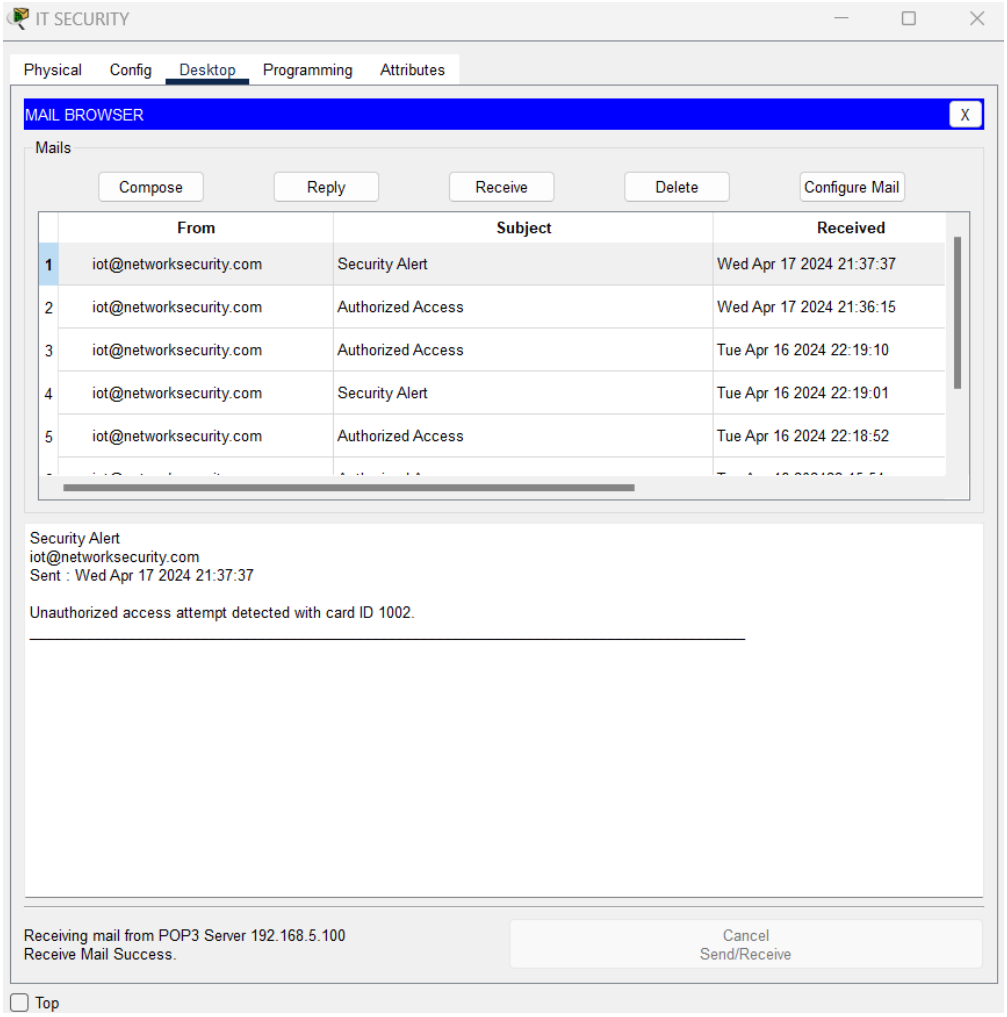


Figure 5.18: Smart ID Card System email alert - II

5.6 Scenario 6: Ransomware Payload Testing

MALICIOUS HOST successfully runs a script that establishes a connection to a RESTRICTED AREA DOOR. Upon establishing a connection, it transmits a command that activates the ransomware attack. After receiving this command, the IoT device executes it and sends a response confirming that the command has been executed, accompanied by a 'Command processed' message. Afterwards, the door emulates the impact of the attack by sending a warning notification, indicating that the system has been compromised and the data has been encrypted, along with instructions for a ransom payment in Bitcoin. This verifies the successful delivery of the payload, shown in **Figure 5.19**

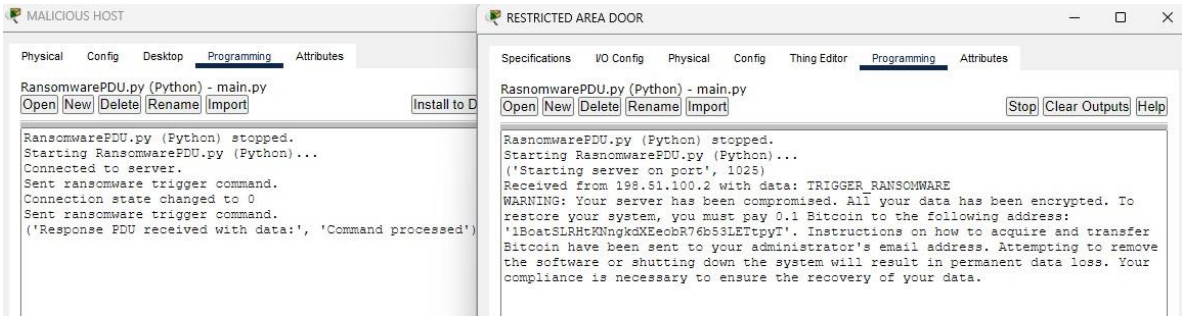


Figure 5.19: Ransomware Payload Success

5.7 Scenario 7: Firewall 1 and 2 Testing Using Ransomware Payload

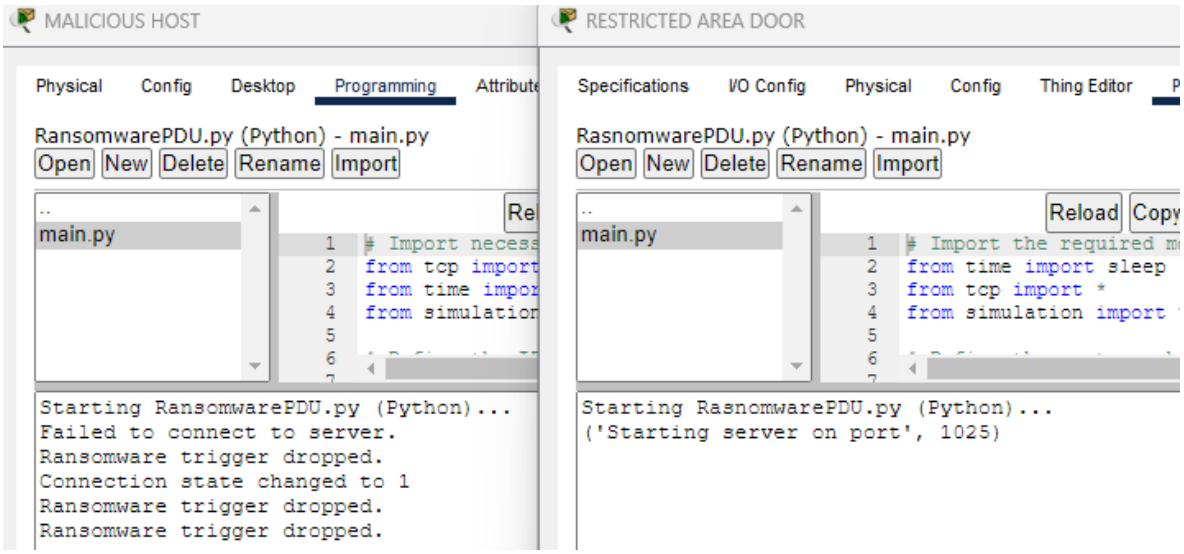


Figure 5.20: Firewall 1 testing against Ransomware Payload

Figure 5.20 highlights the performance of Firewall 1 in a simulated network setting, where a malicious host attempts to set up a connection and execute a ransomware payload. The firewall immediately blocks the connection attempt, providing strong external defence systems that prevent the ransomware trigger from compromising the IoT device. The outcome highlights the firewall's function in detecting and mitigating potential threats, accordingly protecting the network against external cyber-attacks.

The final result in **Figure 5.21** shows that similar to Firewall 1, Firewall 2 successfully blocked the ransomware payload, as indicated by the "Failed to connect to server" notification. This demonstrates that Firewall 2 has strong security strategies, preventing ransomware from accessing or spreading within the network. Both firewalls are crucial to the network's defence plan against cyber-attacks.

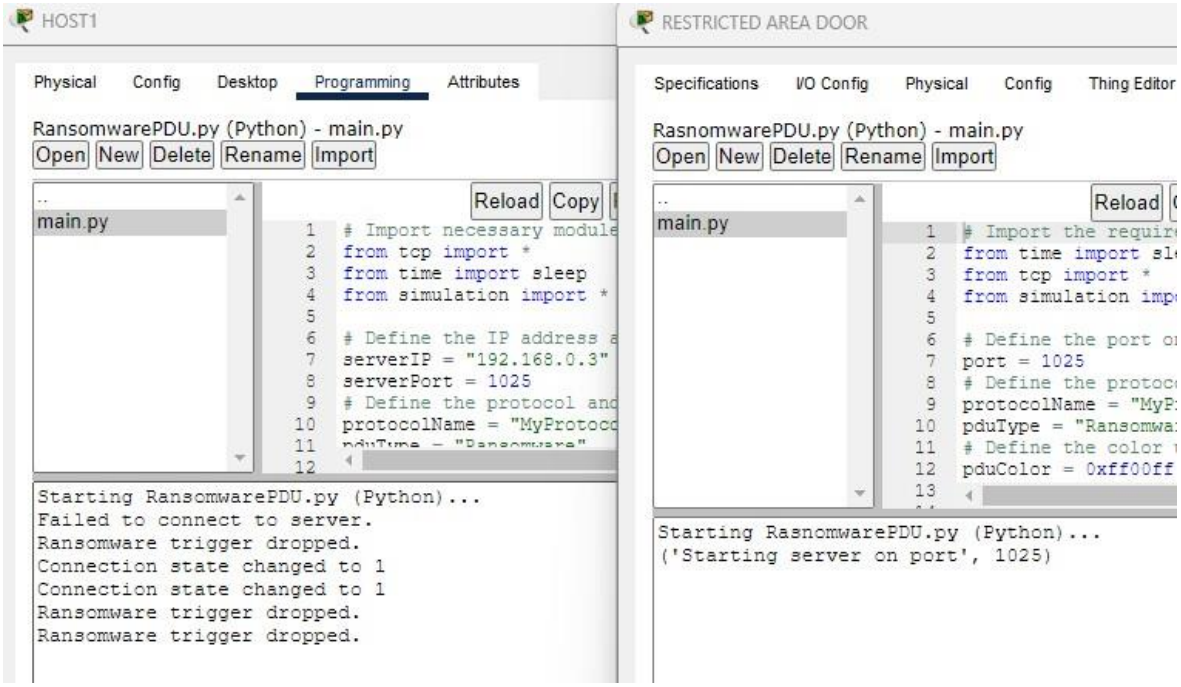


Figure 5.21: Firewall 2 testing against Ransomware Payload

5.8 Scenario 8: SDN Testing

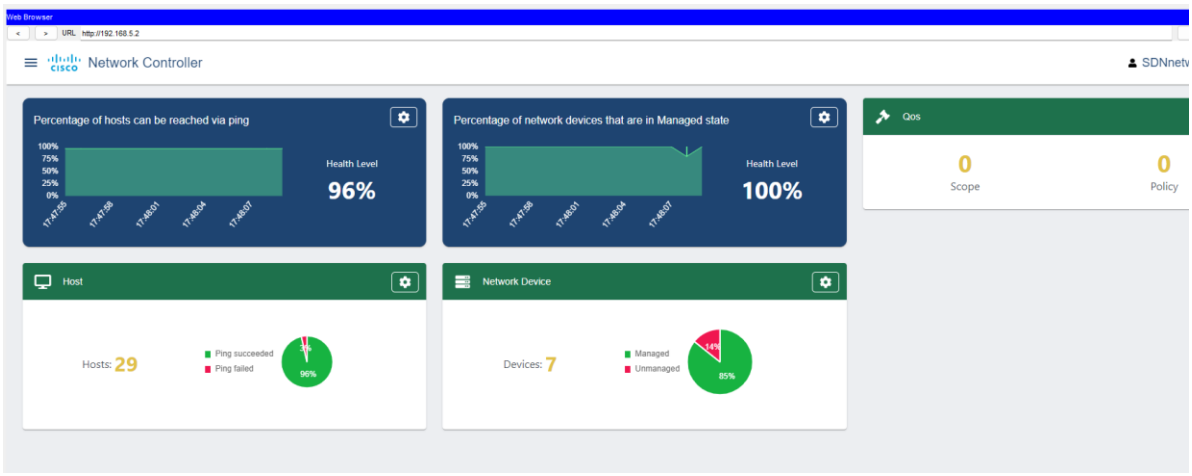


Figure 5.22: SDN dashboard

The dashboard provides an overview of the SBMS network's status and connectivity in the SDN controller. It displays the percentage of hosts that can be reached through ICMP ping requests, directly indicating network connectivity and health. Another such statistic is the ratio of network devices that are in a managed state, indicating the level of supervision and management that network administrators have over the devices.

Figure 5.23 displays the network's healthy condition, with a 100% ability to connect to hosts and complete control over network devices, emphasising the system's exceptional performance and availability.

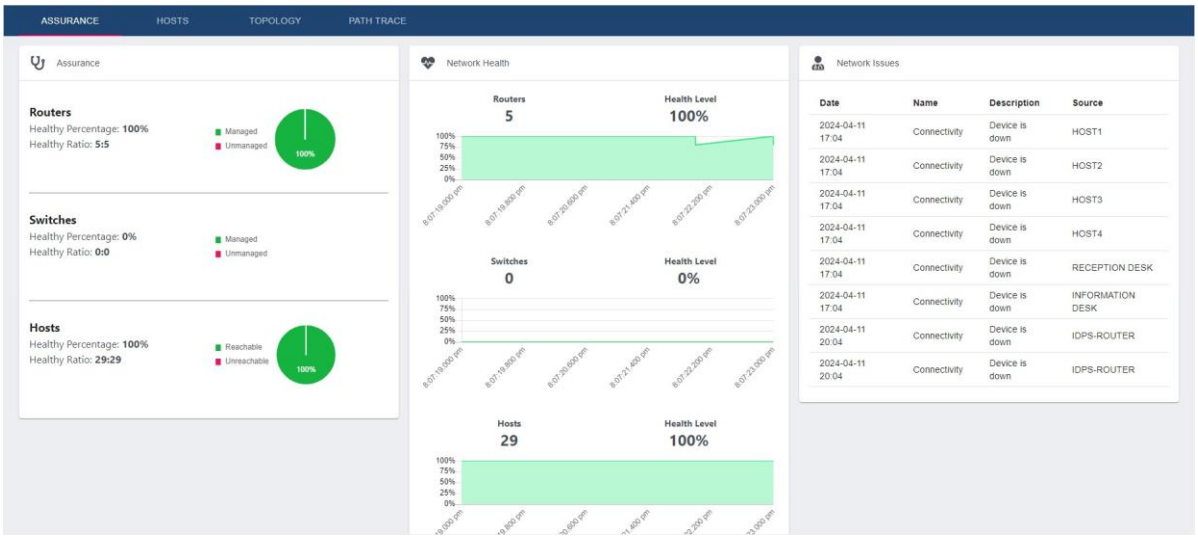


Figure 5.23: SDN Assurance dashboard

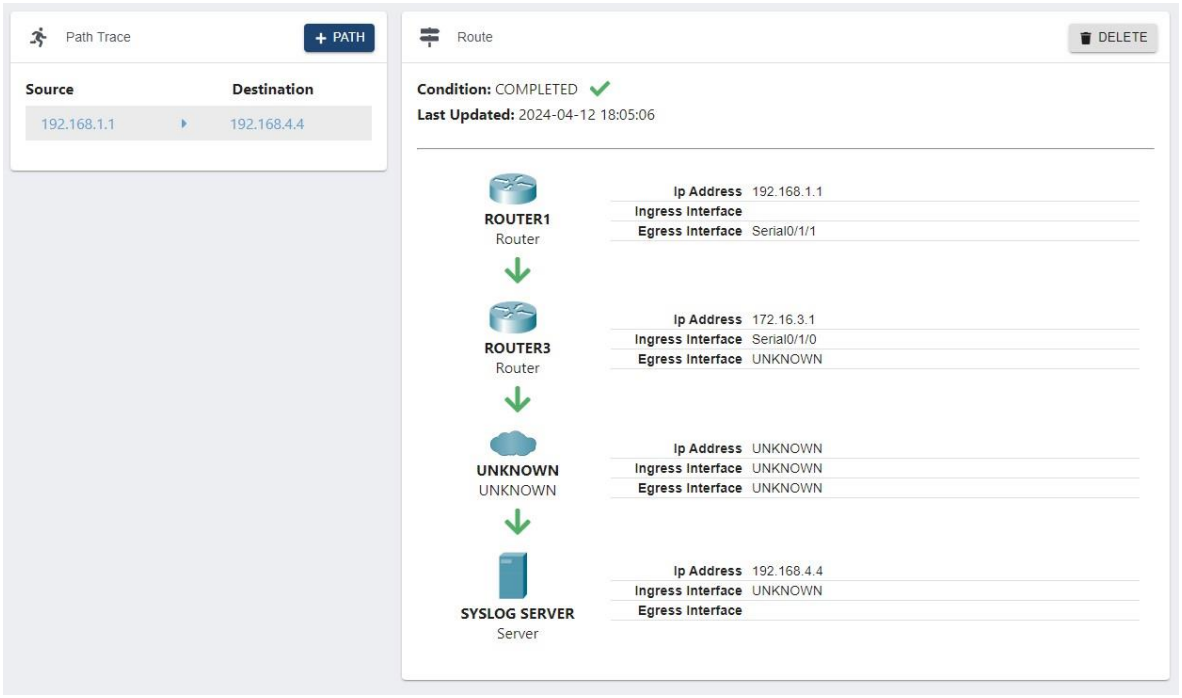


Figure 5.24: SDN Path Trace

Figure 5.24 provides the Path Trace feature, which allows for monitoring and inspecting the network path between two specific points. In this case, it shows the path from a source IP address to a destination IP. The trace displays every step along the route, including the routers and interfaces the traffic goes through. The appearance of a checkmark in the completed condition shows a successful path tracing, while the time stamp offers the most recent update, enabling the network administrator to verify the current state and real-time connectivity. This tool is essential for detecting network faults, enhancing performance, and maintaining security compliance by understanding the data flow within the network system.

5.9 Scenario 9: ServiceTicket.py Testing

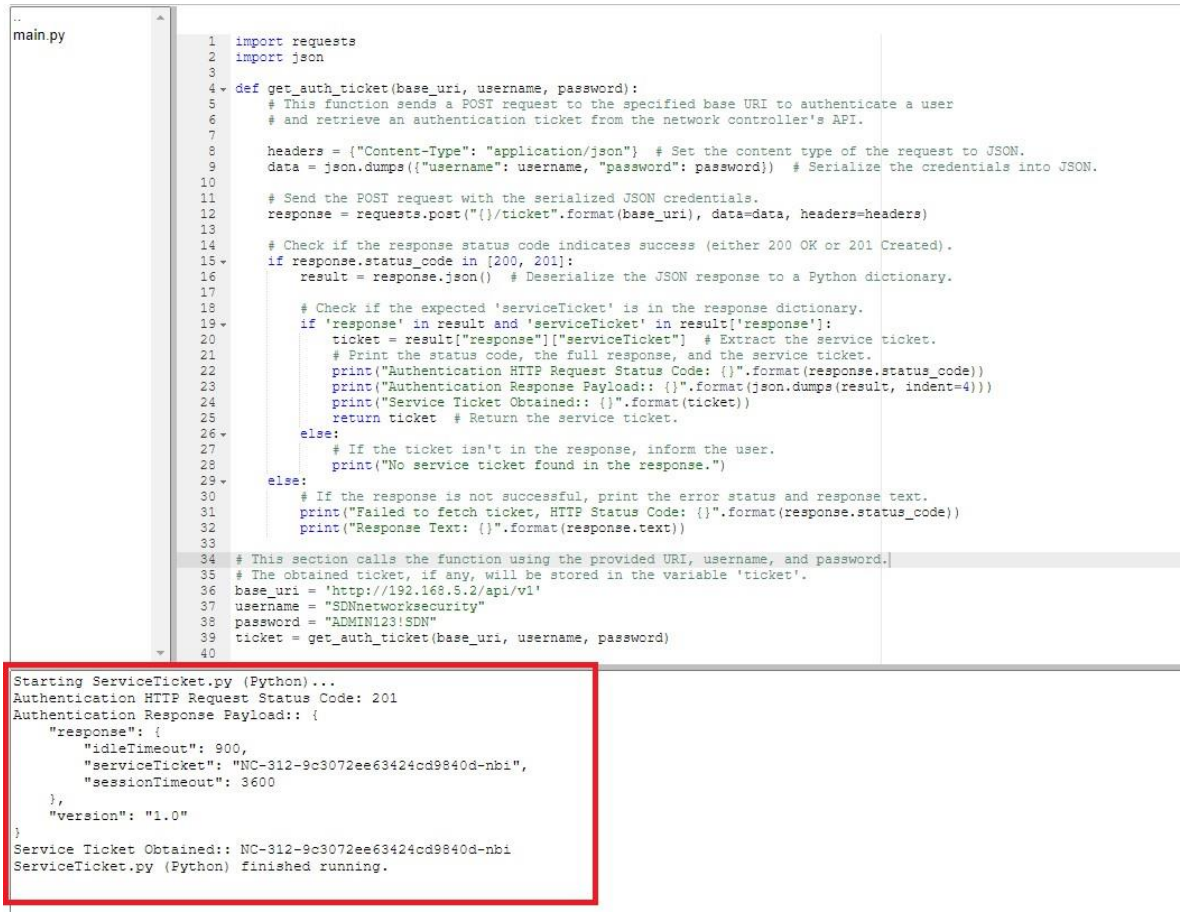


Figure 5.25: Service Ticket.py

Figure 5.25 shows that the script has successfully obtained a service ticket. The output, which includes a 201 HTTP status code and the serviceTicket value, confirms that the credentials are valid and that the user is now authenticated to use the SDN controller's features for monitoring network security.

5.10 Scenario 10: NetworkDevice.py Testing

The networkdevice.py, obtains data about network devices by using an authenticated service ticket. The get_network_devices method uses a GET request to submit a service ticket to the network-device API end-point, included in the header. After receiving the JSON response, which includes information about network devices, it is then returned. The print_devices_info function sequentially processes the received device information and displays each device's hostname, serial number, and software version, expanding the network's inventory visibility.

```
}
------------------------------------------------------------------
Device Information:
------------------------------------------------------------------
HOSTNAME - IDPS-ROUTER, SERIAL NUMBER - FDO1302D6CK-, SOFTWARE VERSION - 15.4
HOSTNAME - ROUTER2, SERIAL NUMBER - FDO13026B95-, SOFTWARE VERSION - 15.4
HOSTNAME - ROUTER3, SERIAL NUMBER - FDO1302I6Q8-, SOFTWARE VERSION - 15.4
HOSTNAME - ROUTER1, SERIAL NUMBER - FDO1302XI7K-, SOFTWARE VERSION - 15.4
HOSTNAME - ISP, SERIAL NUMBER - FDO130213SR-, SOFTWARE VERSION - 15.4
NetworkDevices.py (Python) finished running.
```

Figure 5.26: Network Device.py

The output is demonstrated in **Figure 5.26** by presenting a list of the hostnames and relevant information for routers and the ISP device, showing their current software versions. These details can be used to investigate if there is any unauthorised devices have access.

5.11 Scenario - 11 Host.py Testing

The host.py script involves obtaining a list of host devices from the network by sending a GET request to the host API endpoint of the network controller. The request header includes a service ticket to ensure that the access is authorised. Afterwards, the response, which is in JSON format and includes specific information about the host devices, is retrieved and analysed. The print_devices_info function aims to continuously go through the information of each host device, displaying important details like the hostname, IP, and MAC address.

```
------------------------------------------------------------------
Device Information:
------------------------------------------------------------------
HOSTNAME - FIRE EXIT DOOR, IP - 192.168.5.3, MAC - 0001.6441.5526
HOSTNAME - WINDOW, IP - 192.168.5.4, MAC - 0007.EC52.44E4
HOSTNAME - FIRE SPRINKLE, IP - 192.168.5.5, MAC - 0002.4A43.8EE7
HOSTNAME - FIRE MONITOR, IP - 192.168.5.6, MAC - 00D0.FFEA.BD08
HOSTNAME - FIRE DETECTED LCD SCREEN, IP - 192.168.5.7, MAC - 00D0.9779.D487
HOSTNAME - IT SECURITY, IP - 192.168.5.8, MAC - 00E0.A3C9.4775
HOSTNAME - MALICIOUS HOST, IP - 198.51.100.2, MAC - 00E0.8F67.1662
HOSTNAME - INFORMATION DESK, IP - 192.168.3.3, MAC - 00D0.BCA2.1289
HOSTNAME - SYSLOG SERVER, IP - 192.168.4.4, MAC - 0010.1142.0965
HOSTNAME - DNS SERVER, IP - 192.168.4.2, MAC - 0090.2124.85D7
HOSTNAME - HOST3, IP - 192.168.2.2, MAC - 0060.2FA9.6D9C
HOSTNAME - HOST4, IP - 192.168.2.3, MAC - 000D.BD75.35A8
HOSTNAME - HOST1, IP - 192.168.1.2, MAC - 00E0.8F38.4B11
HOSTNAME - HOST2, IP - 192.168.1.3, MAC - 0002.16AA.05C8
HOSTNAME - RECEPTION DESK, IP - 192.168.3.2, MAC - 0001.4302.7BBB
Host.py (Python) finished running.
```

Figure 5.27: Host.py

5.12 Scenario 12: IDPS Testing using Ransomware Payload

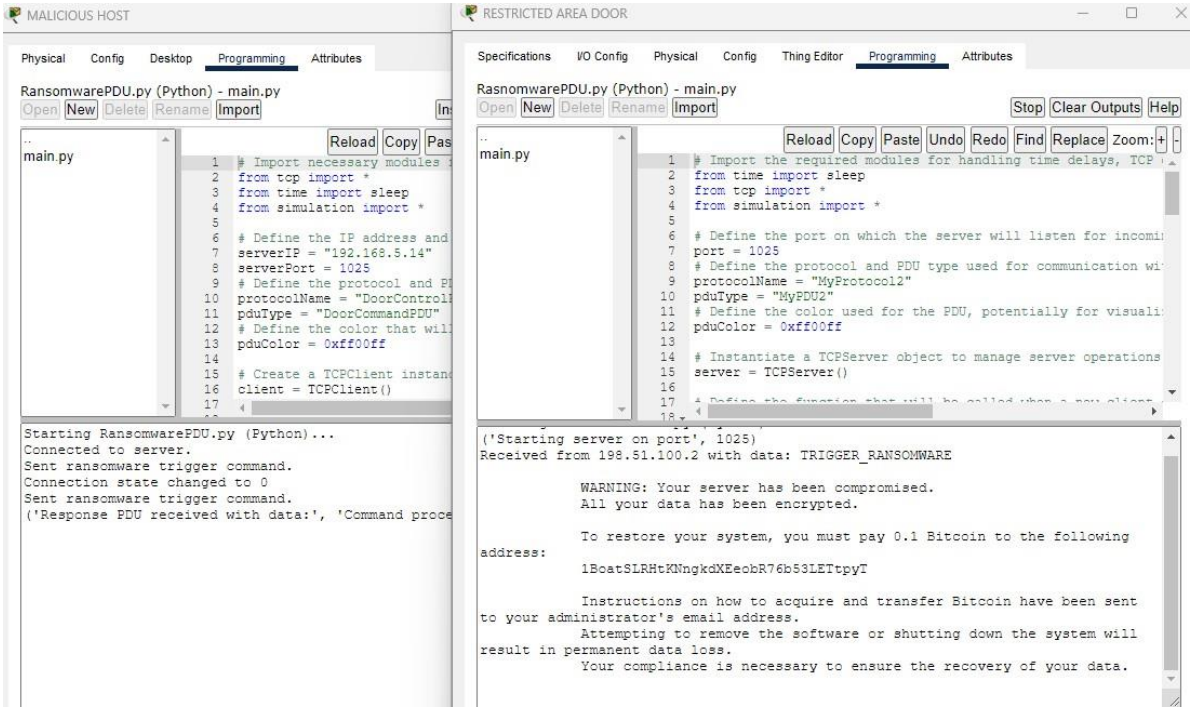


Figure 5.28: IDPS Testing using Ransomware Payload

This IDPS test simulation demonstrates a ransomware attack attempt. The malicious host transmits a command, and the IoT recognises it, indicating malicious encryption of the system. As a result of the limitations of the Cisco Packet Tracer, the simulation is unable to accurately replicate the signature-based detection of an Intrusion Detection and Prevention System (IDPS), leading to a simulated IoT device encryption. This highlights the need for more advanced testing methods to ensure accurate Intrusion Detection and Prevention Systems (IDPS) validation.

5.13 Scenario 12: IDPS Testing using ICMP

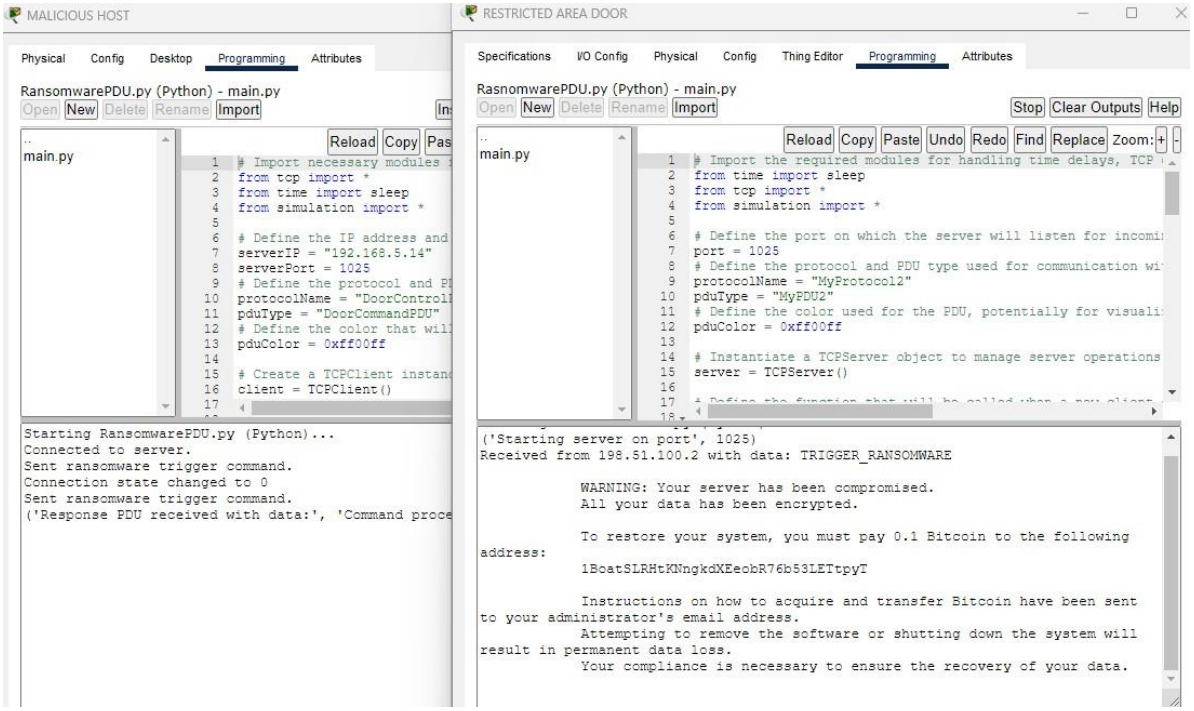


Figure 5.29: IDPS Testing using Ransomware Payload

The PDU List Window displays the outcomes of ICMP ping requests made between different devices. The SDN CONTROLLER effectively transmits a ping to the MALICIOUS HOST, confirming that outbound ICMP requests are permitted. Nevertheless, MALICIOUS HOST's attempts to ping the SDN CONTROLLER and the RESTRICTED AREA DOOR are unsuccessful. This indicates that the incoming ICMP queries from the MALICIOUS HOST are being obstructed, so effectively prohibiting the malicious host from gaining access to the IoT network.

5.14 Scenario 13: Multi-Layered Security Framework Testing using Ransomware Payload

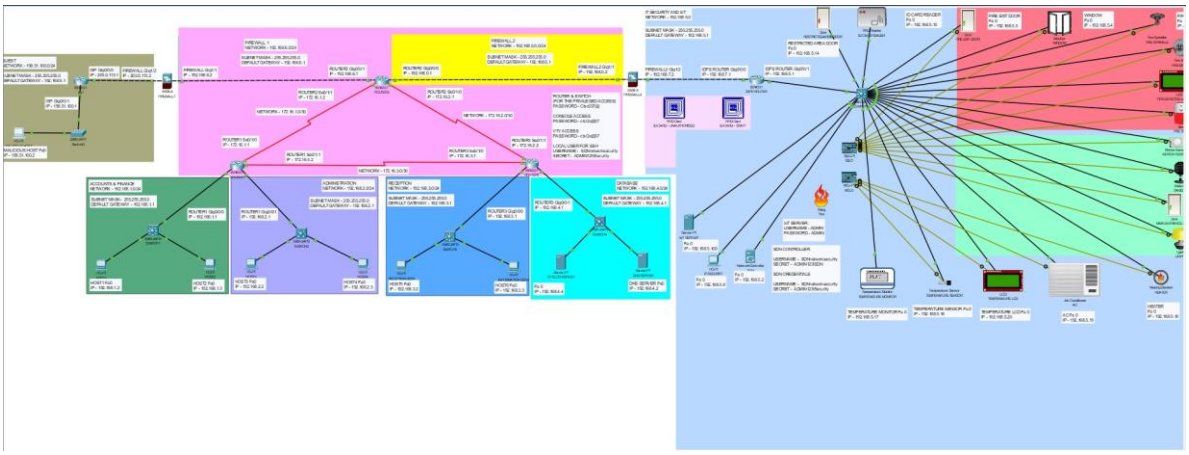


Figure 5.30: Multi-Layered Security Framework

A multi-layered approach is crucial in creating resilient systems that can withstand various threats, provide redundancy, and maintain service integrity. It is commonly used in complex settings, including as distributed systems, cloud computing, and IoT networks, where the interaction between components is complex and there are many opportunities for potential attacks.

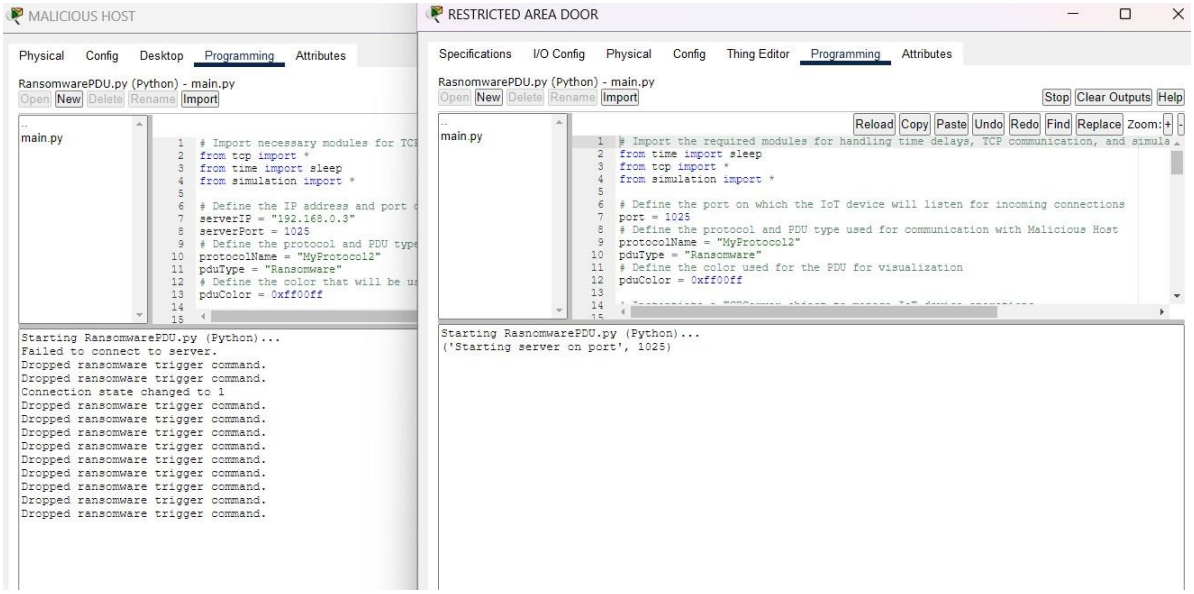


Figure 5.31: Multi-Layered Security Framework Testing using Ransomware Payload

The robust multi-layered safety system successfully stopped the ransomware attack and declared it safe. This defence method applied a comprehensive approach that included multiple layers of defensive measures, each specifically tailored to stopping any threats that may have escaped the preceding layers. The attack was effectively neutralised, demonstrating the strength and effectiveness of the multi-layered security approach in countering harmful cyber attacks.

5.15 Simulation Testing Table

Test ID	Test Description	Expected Outcome	Actual Outcome	Pass/Fail
1	Network Connectivity	Ping Successful	Ping Successful	Pass
2	Smart Fire Alert System	Successfully working	Successfully working	Pass
3	Smart Motion Detection System	Successfully working	Successfully working	Pass
4	Smart Temperature Control System	Successfully working	Successfully working	Pass
5	Smart ID Card Reader	Successfully working	Successfully working	Pass
6	Ransomware Payload	Successfully sent	Successfully sent	Pass
7	Firewall 1 & 2	Detects Ransomware	Detects Ransomware	Pass
8	SDN	Successfully working	Successfully working	Pass
9	ServiceTicket.py	Successfully working	Successfully working	Pass
10	NetworkDevice.py	Successfully working	Successfully working	Pass
11	Host.py	Successfully working	Successfully working	Pass
12	IDPS - Ransomware Payload	Detect Ransomware	Did not Detect	Fail
13	IDPS - ICMP	Successfully blocks	Successfully blocks	Pass
14	Multi-Layered Security	Detects Ransomware	Detects Ransomware	Pass

Table 5.1: Simulation Testing Table

5.16 Limitation of using Packet Tracer

Packet Tracer played a crucial role in generating the network environment in the Smart Business Management System (SBMS) project, although it had severe limitations:

- **Malware Simulation Constraints**: Packet Tracer does not have built-in capabilities for directly simulating malware behaviour. In order to tackle this issue, a TCP client-server model was used to simulate ransomware attack situations. However, it should be noted that this method could not fully replicate the complex nature of actual ransomware interactions.
- **IDPS Functional Limits**: The tool cannot create or modify IDPS signatures, which restricts the ability to evaluate the effectiveness of responses to emerging security risks and the evaluation of the system's adaptability.
- SDN Monitoring and Policy Enforcement Issues: Limitations exist in the SDN capabilities of Packet Tracer, specifically in the areas of monitoring and policy enforcement. These limitations restrict the ability to simulate network control and threat mitigation accurately.

Although Packet Tracer has several limitations, its user-friendly interface and powerful capabilities offer essential insights into network design and effectively explain complex network settings.

”Even the bravest cyber defence will experience defeat when weaknesses are neglected.”

— Stephane Nappo

Chapter 6

EVALUATION

The objectives, as stated in the early sections, were to establish an advanced security framework using firewalls, Software-Defined Networking (SDN), and Intrusion Detection and Prevention Systems (IDPS). The design and implementation of each component were carefully documented in chapters 2 to 4. The effectiveness of these objectives was assessed by comprehensive simulation testing, which showed that the combination of advanced security components created a strong defence against a series of simulated ransomware attacks.

The research adopted a simulation-based approach, using Cisco Packet Tracer to model an SBMS IoT network. The methodology, as explained in Chapter 3, provided an in-depth study of network behaviours during ransomware attacks and a detailed assessment of the security framework’s performance. The simulations confirmed the endurance of the system, but they also revealed several inherent limits in simulated situations, as highlighted in Chapter 5. However, the dependence on a simulated configuration, as compared to a real-life

setting, might be considered both an advantage and a disadvantage. While it provides a secure environment for testing, it may not fully reproduce the unpredictable nature of actual network traffic and ransomware actions.

The study underlined notable features of the project, particularly the efficient identification and countermeasures against ransomware vectors. The techniques were based on the literature review, which provided a comprehensive analysis of the present challenges and resolutions in IoT network security. The study effectively demonstrated a practical understanding of these issues, offering a novel viewpoint to the literature on cyber defence.

Meanwhile, it is important to recognise that there were some weaknesses. The project's scope constrained the investigation of the psychological consequences of ransomware on stakeholders inside an organisation, which is a topic that requires additional research. Furthermore, the project failed to consider the dynamic nature of ransomware attacks that might potentially bypass existing detection methods. This highlights a possible area for future improvement in the security architecture.

Ultimately, the project successfully achieved its primary goals by delivering a methodically planned and successfully implemented security solution for preventing ransomware threats in IoT networks. The project's strengths, including its solid theoretical foundation, thorough simulation testing, and comprehensive security strategy, substantially surpass its limitations. Hence, it can be confidently stated that this dissertation makes a valuable contribution to the field of cybersecurity, particularly in the context of IoT-enabled smart business systems.

6.1 Future Work

Although this dissertation effectively developed and evaluated a strong framework for identifying and reducing the impact of ransomware in IoT networks of Smart Business Management Systems (SBMS), there are several opportunities for additional research and improvement of the project. The utilisation of Cisco Packet Tracer's simulated environment facilitated the analysis of the suggested security framework in a controlled setting, hence improving its real-world application. Nevertheless, IoT networks in the real world demonstrate increased complexity and unpredictability. Future efforts should focus on implementing the security architecture in a live SBMS IoT environment. This would provide valuable insights into the framework's performance under real-world operational conditions and identify new challenges and opportunities for improvement. The existing Intrusion Detection and Prevention System (IDPS) used in the project has limitations in its capability to identify ransomware in encrypted network traffic. Subsequent versions could prioritise improving the detection skills by employing machine learning methods that can determine malicious motives from metadata and traffic patterns without requiring decryption.

Bibliography

Abdullahi, Mujaheed et al. (2022). "Detecting cybersecurity attacks in internet of things using artificial intelligence methods: A systematic literature review". In: *Electronics* 11.2, p. 198.

Akbanov, Maxat, Vassilios G Vassilakis, and Michael D Logothetis (2019). "Ransomware detection and mitigation using software-defined networking: The case of WannaCry". In: *Computers & Electrical Engineering* 76, pp. 111–121.

Alotaibi, Fahad M and Vassilios G Vassilakis (2021). "Sdn-based detection of self-propagating ransomware: the case of badrabbit". In: *Ieee Access* 9, pp. 28039–28058.

Alwashali, Ali Ahmed Mohammed Ali, Nor Azlina Abd Rahman, and Noris Ismail (2021). "A survey of ransomware as a service (RaaS) and methods to mitigate the attack". In: *2021 14th International Conference on Developments in eSystems Engineering (DeSE)*. IEEE, pp. 92–96.

Anghel, Mihail and Andrei Racautanu (2019). *A note on different types of ransomware attacks*. Cryptology ePrint Archive, Paper 2019/605. https : / / eprint . iacr . org / 2019 / 605. URL: https : / / eprint.iacr.org/2019/605.

Beaman, Craig et al. (2021). "Ransomware: Recent advances, analysis, challenges and future research directions". In: *Computers & security* 111, p. 102490.

Beigh, Bilal Maqbool, Uzair Bashir, and Manzoor Chahcoo (2013). "Intrusion detection and prevention system: issues and challenges". In: *International Journal of Computer Applications* 76.17.

Cabaj, Krzysztof, Marcin Gregorczyk, and Wojciech Mazurczyk (2018). "Software-defined networking-based crypto ransomware detection using HTTP traffic characteristics". In: *Computers & Electrical Engineering* 66, pp. 353–368.

Coulibaly, Keturahlee (2020). "An overview of intrusion detection and prevention systems". In: *arXiv preprint arXiv:2004.08967*.

Dastres, Roza and Mohsen Soori (2021). "A review in recent development of network threats and security measures". In: *International Journal of Information Sciences and Computer Engineering*.

Ekta and Urvashi Bansal (2021). "A Review on Ransomware Attack". In: *2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC)*, pp. 221–226. DOI: 10.1109/ICSCCC51823.2021.9478148.

Elgazzar, Khalid et al. (2022). "Revisiting the internet of things: New trends, opportunities and grand challenges". In: *Frontiers in the Internet of Things* 1, p. 1073780.

Elsayed, Rania A et al. (2023). "Securing IoT and SDN systems using deep-learning based automatic intrusion detection". In: *Ain Shams Engineering Journal* 14.10, p. 102211.

Fernando, Damien Warren, Nikos Komninos, and Thomas Chen (2020). "A study on the evolution of ransomware detection using machine learning and deep learning techniques". In: *IoT* 1.2, pp. 551–604.

Fritts, Brett (2021). "Firewall: Configuration Deficiencies". PhD thesis. Utica College.

Gerodimos, Apostolos et al. (2023). "IoT: Communication protocols and security threats". In: *Internet of Things and Cyber-Physical Systems* 3, pp. 1–13. ISSN: 2667-3452. DOI: https://doi.org/10.1016/j.iotcps. 2022 . 12 . 003. URL: https://www.sciencedirect.com/science/article/pii/S2667345222000293.

Al-Hawawreh, Muna, Elena Sitnikova, and Neda Aboutorab (2021). "Asynchronous peer-to-peer federated capability-based targeted ransomware detection model for industrial iot". In: *IEEE Access* 9, pp. 148738–148755.

Humayun, Mamoona et al. (2021). "Internet of things and ransomware: Evolution, mitigation and prevention". In: *Egyptian Informatics Journal* 22.1, pp. 105–117.

Kapoor, Adhirath et al. (2021). "Ransomware detection, avoidance, and mitigation scheme: a review and future directions". In: *Sustainability* 14.1, p. 8.

Kara, Ilker and Murat Aydos (2022). "The rise of ransomware: Forensic analysis for windows based ransomware attacks". In: *Expert Systems with Applications* 190, p. 116198.

Karmakar, Kallol Krishna et al. (2020). “SDN-enabled secure IoT architecture”. In: *IEEE Internet of Things Journal* 8.8, pp. 6549–6564.

Laghari, Asif Ali et al. (2021). “A review and state of art of Internet of Things (IoT)”. In: *Archives of Computational Methods in Engineering*, pp. 1–19.

Lund, Ryan, Anthony Fenzl, and Chelsea Villanueva (2020). “Distributed firewall for iot”. In.

Maleh, Yassine et al. (2023). “A comprehensive survey on SDN security: threats, mitigations, and future directions”. In: *Journal of Reliable Intelligent Environments* 9.2, pp. 201–239.

McIntosh, Timothy et al. (2021). “Ransomware mitigation in the modern era: A comprehensive review, research challenges, and future directions”. In: *ACM Computing Surveys (CSUR)* 54.9, pp. 1–36.

Mofidi, Farhad, Sena Hounsinou, and Gedare Bloom (2023). “L-IDS: A lightweight hardware-assisted IDS for IoT systems to detect ransomware attacks”. In: *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation*, pp. 464–465.

Mukkamala, Padma Priya and Sindhu Rajendran (2020). “A survey on the different firewall technologies”. In: *International Journal of Engineering Applied Sciences and Technology* 5.1, pp. 363–365.

Nizˇetic´, Sandro et al. (2020). “Internet of Things (IoT): Opportunities, issues and challenges towards a smart and sustainable future”. In: *Journal of cleaner production* 274, p. 122877.

Oloyede, OA et al. (2021). “Firewall Approach to Computer Network Security: Functional Viewpoint”. In: *International Journal of Advanced Networking and Applications* 13.3, pp. 4993–5000.

Paga´n, Alexander and Khaled Elleithy (2021). “A multi-layered defense approach to safeguard against ransomware”. In: *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, pp. 0942–0947.

Ramadan, Rabie A and Kusum Yadav (2020). “A novel hybrid intrusion detection system (IDS) for the detection of internet of things (IoT) network attacks”. In: *Annals of Emerging Technologies in Computing (AETiC), Print ISSN*, pp. 2516–0281.

Saeed, Soobia et al. (2020). “Ransomware: A framework for security challenges in internet of things”. In: *2020 2nd International Conference on Computer and Information Sciences (ICCIS)*. IEEE, pp. 1–6.

Scott-Hayward, Sandra, Gemma O’Callaghan, and Sakir Sezer (2013). “SDN security: A survey”. In: *2013 IEEE SDN For Future Networks and Services (SDN4FNS)*. IEEE, pp. 1–7.

Teichmann, Fabian, Sonia R Boticiu, and Bruno S Sergi (2023). “The evolution of ransomware attacks in light of recent cyber threats. How can geopolitical conflicts influence the cyber climate?” In: *International Cybersecurity Law Review* 4.3, pp. 259–280.

Thapa, Suman and Akalanka Mailewa (2020). “The role of intrusion detection/prevention systems in modern computer networks: A review”. In: *Conference: Midwest Instruction and Computing Symposium (MICS)*. Vol. 53, pp. 1–14.

Wani, Azka and S Revathi (2020). “Ransomware protection in IoT using software defined networking”. In: *Int. J. Electr. Comput. Eng* 10.3, pp. 3166–3175.

Warikoo, Arun (2023). “Perspective Chapter: Ransomware”. In: *Malware*. Ed. by Eduard Babulak. Rijeka: IntechOpen. Chap. 5. DOI: 10.5772/intechopen.108433. URL: https://doi.org/10.5772/intechopen.108433.

Wheelus, Charles and Xingquan Zhu (2020). “IoT network security: Threats, risks, and a data-driven defense framework”. In: *IoT* 1.2, pp. 259–285.

Wu, Dafei (2021). “Research on Network Security Defense Model Based on Combination Strategy of Firewall and IPS”. In: *Forest Chemicals Review*, pp. 324–331.

Yamany, Bahaa et al. (2022). “A New Scheme for Ransomware Classification and Clustering Using Static Features”. In: *Electronics* 11.20, p. 3307.

Appendix A

APPENDIX

A.1 Ransomware Payload - Malicious Host (CLIENT TCP)

```
×
# Import necessary modules for TCP communication and delaying operations
from tcp import *
from time import sleep
from simulation import *

# Define the IP address and port of the IoT device that Malicious Host will connect to
serverIP = "192.168.0.3"
serverPort = 1025
# Define the protocol and PDU type that will be used for sending data
protocolName = "MyProtocol2"
pduType = "Ransomware"
# Define the color that will be used for the PDU for visualization purposes
pduColor = 0xff00ff

# Create a TCPClient instance for managing the TCP connection
client = TCPClient()

# Callback function that's called when the TCP connection state changes
def onTCPConnectionChange(type):
    # Print the new connection state to the console
    print ("Connection state changed to {}".format(type))

# Callback function for handling PDU received from the IoT device
def onTCPReceiveWithPDUInfo(data, pduInfo):
    # Print the response data from the IoT device to the console
    print ("Response PDU received with data :", data)

# The main function where the Malicious Host's behavior is defined
def main():
    # Register the callback functions with the Malicious Host instance
```

```
        client .onConnectionChange(onTCPConnectionChange)
        client .onReceiveWithPDUInfo(onTCPReceiveWithPDUInfo)

    # Attempt to connect to the server using the provided IP address and port
    if client .connect( serverIP , serverPort ):
        # If the connection is successful , print a confirmation message
        print (”Connected to server .”)
    else :
        # If the connection fails , print an error message and exit the function
        print (”Failed to connect to server .”)
        return

    # Enter an infinite loop where the Malicious Host periodically sends PDUs to the IoT device
    while True:
        # Create a PDUInfo object with the specified color
        pduInfo = PDUInfo(pduColor)
        # Add a message to the PDU
        pduInfo.addOutMessage(”Ransomware trigger command”)

        # Set the format of the outgoing PDU with the command to trigger ransomware
        pduInfo.setOutFormat(protocolName, pduType, {
            ”type”: ”COMMAND”,
            ”command”: ”TRIGGER_RANSOMWARE”
        })
        # Send the PDU to the IoT device
        client .sendWithPDUInfo(”TRIGGER_RANSOMWARE”, pduInfo)
        # Print a message to the console confirming that the ransomware trigger was sent
        print (”Sent ransomware trigger command.”)
        # Pause the loop for 10 seconds before sending the next PDU
        sleep (10)

# Check if the script is run as the main module and, if so, call the main function
if __name__ == ”__main__”:
    main()
```

A.2 Ransomware Payload - Iot Device (Server TCP)

```
×
# Import the required modules for handling time delays, TCP communication, and simulation-specific tasks
from time import sleep
from tcp import *
from simulation import *

# Define the port on which the IoT device will listen for incoming connections
port = 1025
# Define the protocol and PDU type used for communication with Malicious Host
```

```
protocolName = "MyProtocol2"
pduType = "Ransomware"
# Define the color used for the PDU for visualization
pduColor = 0xff00ff

# Instantiate a TCPServer object to manage IoT device operations
server = TCPServer()

# Define the function that will be called when Malicious Hostt connects to the server
def onTCPNewClient(client):
    # Define a nested function for handling changes in the TCP connection state
    def onTCPConnectionChange(type):
        # When the connection state changes, print a message with the Malicious Host's remote IP and the new state
        print("Connection to " + client.remoteIP() + " changed to state " + str(type))

    # Define a nested function to handle data received from the Malicious Host, along with PDU information
    def onTCPReceiveWithPDUInfo(data, pduInfo):
        # Print a message displaying the Malicious Host's IP and the data received
        print("Received from {} with data: {}".format(client.remoteIP(), data))
        # Check if the received data is a "trigger_ransomware" command
        if data.strip().lower() == "trigger_ransomware":
            # If the command is received, print a simulated ransomware message
            print("""
            WARNING: Your server has been compromised.
            All your data has been encrypted.

            To restore your system, you must pay 0.1 Bitcoin to the following address:
            1BoatSLRHtKNngkdXEeobR76b53LETtpyT

            Instructions on how to acquire and transfer Bitcoin have been sent to your administrator's email address.
            Attempting to remove the software or shutting down the system will result in permanent data loss.
            Your compliance is necessary to ensure the recovery of your data.
            """)
        # Add messages to the PDU information acknowledging reception and processing of the command
        pduInfo.addInMessage("Received your command.")
        pduInfo.addOutMessage("Processed your command.")
        # Set the PDU format for the reply to indicate that the command was processed
        pduInfo.setOutFormat(protocolName, pduType, {"type": "REPLY", "data": "Command processed"})
        # Send a response back to the Malicious Host with the updated PDU information
        client.sendWithPDUInfo("Command processed", pduInfo)

    # Register the above nested functions as callbacks to handle events for each client
    client.onConnectionChange(onTCPConnectionChange)
    client.onReceiveWithPDUInfo(onTCPReceiveWithPDUInfo)

# Define a setup function to initialize the server and configure PDUs
def setup():
```

```
56        # Add a custom PDU format to the simulation with specific fields and properties
57        Simulation.addCustomPDU(protocolName, pduType, {
58            "title": "My PDU2",
59            "units": "Bits",
60            "unit_marks": [16],
61            "width": 32,
62            "fields": [
63                {"value": "TYPE: {type}", "size": 32},
64                {"value": "DATA: {data}", "size": 32}
65            ]
66        })
67        # Print a message indicating that the server is starting and on which port it's listening
68        print("Starting server on port", port)
69        # Register the function to handle Malicious Host connections
70        server.onNewClient(onTCPNewClient)
71        # Start listening for incoming connections on the specified port
72        server.listen(port)
73
74    # Call the setup function to start the IoT Device server
75    if __name__ == "__main__":
76        setup()
77        # Enter an infinite loop to keep the IoT Device server running, checking for new connections or data every 10 seconds
78        while True:
79            sleep(10)
```

Appendix B

APPENDIX

B.1 Smart ID Card Reader

```
×
1   # Import various modules needed for the simulation and control of the RFID reader and email notifications
2   from options import Options
3   from time import sleep
4   from physical import *
5   from gpio import *
```

```
from ioeclient import IoEClient
from email import * # Assume EmailClient is defined elsewhere as shown earlier

# Set constants for the delay between loops and the read distances for the RFID reader
delay_time = 1 # Delay time in seconds
x_read_distance = 50
y_read_distance = 50

# Initialize variables to store the current and last read card IDs, and the current state of the reader
cardID = 0
lastCardID = 0
state = 2 # waiting

# Define an authorized card ID for access control
AUTHORIZED_CARD_ID = 1001

# The setup function configures the IoE (Internet of Everything) client for the RFID reader
def setup():
    IoEClient.setup({
        "type": "RFID Reader",
        "states": [
            {"name": "Card ID", "type": "number", "unit": '', "controllable": False},
            {"name": "Status", "type": "options", "options": {"0": "Valid", "1": "Invalid", "2": "Waiting"}, "controllable": True}
        ]
    })
    # Set up a callback to process data when it is received by the IoE client
    IoEClient.onInputReceive(lambda rinput: processData(rinput, True))
    # Configure the EmailClient with the credentials for sending emails
    EmailClient.setup("iot@networksecurity.com", "networksecurity.com", "IoT", "iOt123!")

# The loop function is the main execution loop of the RFID reader
def loop():
    global cardID, lastCardID, state # Declare globals to modify them within the function
    # Scan for devices within the specified range
    devices = devicesAt(getCenterX(), getCenterY(), x_read_distance, y_read_distance)
    found = False # A flag to check if a valid card is found
    for device in devices:
        if device == getName(): # Skip if the detected device is the reader itself
            continue

        # Try to get the card ID property from the detected device
        tempCardID = getDeviceProperty(device, 'CardID')
        if tempCardID:
            try:
                tempCardID = int(tempCardID) # Convert to an integer
                cardID = tempCardID # Set the card ID
                found = True # Mark that we've found a card
```

```
                    break
                except ValueError:
                    continue  # Ignore the device if the card ID is not an integer

    # If no valid card is found, reset the card ID and set the reader's state to waiting
    if not found:
        cardID = lastCardID = 0
        setState(2)
    else:
        # If a new card ID is read that is different from the last, process it
        if lastCardID != cardID:
            lastCardID = cardID
            if cardID == AUTHORIZED_CARD_ID:
                # If the card is authorized, set the state to valid and print a message
                setState(0)
                print("Mr. John with card ID {} has been granted access to the IT server room.".format(cardID))
                # Send an email notification for authorized access
                EmailClient.send("it@networksecurity.com", "Authorized Access", "Mr. John with card ID {} has been granted access to the IT server room.".format(cardID))
            else:
                # If the card is not authorized, set the state to invalid and print a message
                setState(1)
                print("Unauthorized access attempt detected with card ID {}.".format(cardID))
                # Send an email alert for unauthorized access
                EmailClient.send("it@networksecurity.com", "Security Alert", "Unauthorized access attempt detected with card ID {}.".format(cardID))
            # Send a status report
            sendReport()

    # Wait for the specified delay time before running the loop again
    sleep(delay_time)

# Set the state of the RFID reader and write the state to an analog pin (for physical simulation)
def setState(newState):
    global state
    if state != newState:
        state = newState
        analogWrite(A1, state)
        sendReport()

# Send a report of the current state and card ID to the IoE server
def sendReport():
    report = str(cardID) + "," + str(state)
    IoEClient.reportStates(report)

# Process data received from the IoE server
def processData(data, bIsRemote):
    if not data:
        return
```

```
100         # Split the received data into parts and try to set the state
101         data = data.split(",")
102         if len(data) > 1:
103             try:
104                 newState = int(data[1])
105                 setState(newState)
106             except ValueError:
107                 pass # Ignore if the data cannot be converted to an integer
108
109 # Execute the setup function and then run the loop indefinitely
110 if __name__ == "__main__":
111     setup()
112     while True:
113         loop()
```

Appendix C

APPENDIX

C.1 Smart Fire Alert System

```
1  # Import specific functions from the GPIO module to interact with the hardware pins
2  from gpio import pinMode, digitalRead, customWrite, INPUT, OUTPUT
3  # Import the time and sleep function from the time module to add delays in the program
4  from time import time, sleep
5
6  # Define constants for the GPIO pin numbers for better readability and maintainability
7  input_pin = 0  # The pin number for the fire sensor input
8  sprinkler_pin = 1  # The pin to control the sprinkler system
9  door_pin = 2  # The pin to control the door (assumed to be a binary open/close system)
10 window_pin = 3  # The pin to control the window
11 lcd_pin = 4  # The pin connected to an LCD display for status messages
12 siren_pin = 5  # The pin to control the siren
13 fire_detected_value = 1023  # The sensor value indicating fire detection
14
15 def fire_alert_system():
```

```
16      # Set up the GPIO pins with the appropriate input or output designation
17      pinMode(input_pin, INPUT) # Set the fire sensor pin as input
18      pinMode( sprinkler_pin , OUTPUT) # Set the sprinkler control pin as output
19      pinMode(door_pin, OUTPUT) # Set the door control pin as output
20      pinMode(window_pin, OUTPUT) # Set the window control pin as output
21      pinMode(lcd_pin, OUTPUT) # Set the LCD display pin as output
22      pinMode(siren_pin , OUTPUT) # Set the siren control pin as output
23
24      # Print a starting message to indicate that the fire alert system is now running
25      print (”FIRE ALERT SYSTEM”)
26       consecutive_no_fire_count = 0 # Initialize a counter to track the number of consecutive no−fire readings
27
28      # Enter an infinite loop to continuously check the fire sensor
29      while True:
30           fire_value = digitalRead ( input_pin ) # Read the value from the fire sensor
31
32          # Check if the sensor value corresponds to the defined fire detection value
33          if fire_value == fire_detected_value :
34               handle_fire_detected () # Call a function to handle fire detection actions
35               consecutive_no_fire_count = 0 # Reset the no−fire count upon fire detection
36          else :
37              # If no fire is detected , increment the counter and call the no−fire handler
38               consecutive_no_fire_count += 1 # Increment the no−fire count
39               print (”Consecutive no−fire count: {}”.format( consecutive_no_fire_count )) # Print the count for logging
    purposes
40               handle_no_fire () # Call a function to handle the no−fire actions
41
42          # If the system detects no fire for 10 consecutive readings , exit the loop
43          if consecutive_no_fire_count >= 10:
44               print (”No fire detected for 10 consecutive iterations − Exiting loop”)
45               break # Break out of the loop to stop the system
46
47          sleep (1) # Wait for 1 second before the next loop iteration to reduce CPU usage
48
49  # Define the function to handle actions when fire is detected
50  def handle_fire_detected ():
51      print (”FIRE DETECTED”) # Print a message indicating fire detection
52      customWrite( sprinkler_pin , ’1’) # Activate the sprinkler system
53      customWrite(door_pin, ’1,0 ’) # Open the doors
54      customWrite(window_pin, ’1’) # Open the windows
55      customWrite( lcd_pin , ’FIRE DETECTED’) # Update the LCD display with the fire detection message
56      customWrite( siren_pin , ’1’) # Activate the siren
57
58      # Begin the evacuation announcement procedure when a fire is detected
59      last_announcement_time = time () # Store the initial time when the announcement loop starts
60      while True: # Start an infinite loop to keep announcing until the fire is no longer detected
61          current_time = time () # Get the current time during each iteration of the loop
62
```

```
63            # Check if 5 seconds have passed since the last announcement
64            if current_time - last_announcement_time >= 5:
65                # Print the evacuation message to the console, which would be equivalent to broadcasting the announcement in a real system
66                print("Attention please: This is an automated safety notification. "
67                      "A fire incident has been detected in the building. "
68                      "Please evacuate immediately through the nearest fire exit. "
69                      "Do not use elevators. Repeat, do not use elevators. "
70                      "Proceed calmly to the closest emergency exit and leave the building. "
71                      "Thank you for your cooperation.")
72
73                # Update the last announcement time to the current time after the announcement
74                last_announcement_time = current_time
75
76            # Check if the fire is still detected
77            if digitalRead(input_pin) != fire_detected_value:
78                break # Exit the while loop if no fire is detected, which ends the evacuation announcements
79            sleep(0.1) # A short sleep to prevent this loop from hogging CPU
80
81    # Define the function to handle actions when no fire is detected
82    def handle_no_fire():
83        customWrite(sprinkler_pin, '0') # Deactivate the sprinkler system
84        customWrite(door_pin, '0,1') # Close the doors
85        customWrite(window_pin, '0') # Close the windows
86        customWrite(lcd_pin, 'ALL IS WELL') # Update the LCD display with an all-clear message
87        customWrite(siren_pin, '0') # Deactivate the siren
88
89    # Main entry point of the script
90    if __name__ == '__main__':
91        fire_alert_system() # Call the fire alert system function to start the system
```

Appendix D

APPENDIX

D.1 Smart Motion Detection System

```
×
from gpio import *
from time import *

# Constants for the GPIO pin numbers and motion detected value
motion_sensor_pin = 6
camera_pin = 7
door_pin = 8
light_pin = 9
motion_detector_value = 1023 # The value indicating motion is detected

def setup_pins():
    """Set up GPIO pins as input or output as required."""
    pinMode(motion_sensor_pin, INPUT)
    pinMode(camera_pin, OUTPUT)
    pinMode(door_pin, OUTPUT)
    pinMode(light_pin, OUTPUT)

def motion_detected_actions():
    """Actions to perform when motion is detected."""
    customWrite(camera_pin, '1') # Activate Camera
    customWrite(door_pin, '1,0') # Unlock/Open Door
    customWrite(light_pin, '2') # Turn on Light
    print("Actions executed: Camera activated, Door opened, Light turned on.")

def no_motion_actions():
    """Actions to perform when no motion is detected."""
    customWrite(camera_pin, '0') # Deactivate Camera
    customWrite(door_pin, '0,0') # Lock/Close Door
    customWrite(light_pin, '0') # Turn off Light
    print("Actions executed: Camera deactivated, Door closed, Light turned off.")

def main():
    print("Starting SMART DOOR SYSTEM BY MOTION SENSOR")
    setup_pins()

    no_motion_count = 0 # Initialize counter for consecutive no-motion readings

    while True:
        if digitalRead(motion_sensor_pin) == motion_detector_value:
            print("Security Alert: Motion has been detected near the reception hall. "
                  "An individual may be approaching. Please verify their identification immediately.")
            no_motion_count = 0 # Reset the no-motion counter
            motion_detected_actions()
        else:
            no_motion_count += 1
            print("No motion detected. Count: {}".format(no_motion_count))
            no_motion_actions()
```

```
48
49                  # Exit the loop if no motion is detected for a predefined number of times
50                  if no_motion_count >= 10:
51                      print ("No motion detected for 10 consecutive checks - Exiting system.")
52                      break
53
54              sleep(1)  # Wait for 1 second before checking again
55
56  if __name__ == '__main__':
57      main()
```

Appendix E

APPENDIX

E.1 Smart Temperature Control System

```
1  # Import specific functions from the GPIO module to interact with the hardware pins
2  from gpio import *
3  # Import the time and sleep function from the time module to add delays in the program
4  from time import time, sleep
5
6  # Constants for the GPIO pin numbers and temperature thresholds
7   temperature_sensor_pin = 0
8   ac_pin = 3
9   heater_pin = 2
10  lcd_pin = 1
11  temp_high_threshold = 520  # The temperature value that triggers the AC
12  temp_low_threshold = 510  # The temperature value that triggers the heater
13
14  def setup_pins():
15      """ Initialize GPIO pins."""
16      pinMode(temperature_sensor_pin, INPUT)
17      pinMode(ac_pin, OUTPUT)
18      pinMode(heater_pin, OUTPUT)
19      pinMode(lcd_pin, OUTPUT)
```

```
20
21  def turn_on_ac():
22      """Activate the air conditioning system and update the LCD."""
23      digitalWrite(ac_pin, HIGH)
24      customWrite(lcd_pin, "AC-ON")
25      print("Air conditioning activated.")
26
27  def turn_on_heater():
28      """Activate the heating system and update the LCD."""
29      digitalWrite(heater_pin, HIGH)
30      customWrite(lcd_pin, "HEATER-ON")
31      print("Heating system activated.")
32
33  def maintain_normal_temp():
34      """Turn off all temperature control systems and update the LCD to indicate normal temperature."""
35      digitalWrite(ac_pin, LOW)
36      digitalWrite(heater_pin, LOW)
37      customWrite(lcd_pin, "NORMAL-TEMP")
38      print("Temperature is normal. All systems standby.")
39
40  def main():
41      print("Starting SMART ROOM TEMPERATURE system")
42      setup_pins()
43
44      while True:
45          # Read temperature from the sensor
46          temp = digitalRead(temperature_sensor_pin)
47          print("Temperature Reading: {}".format(temp))
48
49          # Determine and apply temperature control actions
50          if temp >= temp_high_threshold:
51              turn_on_ac()
52          elif temp < temp_low_threshold:
53              turn_on_heater()
54          else:
55              maintain_normal_temp()
56
57          sleep(3)  # Wait for 3 seconds before the next read
58
59  if __name__ == "__main__":
60      main()
```

Appendix F

APPENDIX

F.1 FIRE.

```
×
1
2  function  setup ()
3  {
4      setDeviceProperty (getName(),  'IR',  900);
5  }
```

Appendix G

APPENDIX

G.1 FIREWALL 1 CLI

```
: Saved
: Written by enable_15 at 00:00:00 UTC Mar 1 1993
: Call-home enabled from prompt by enable_15 at 00:00:00 UTC Mar 1 1993
:
ASA Version 9.6(1)
!
hostname FIREWALL1
```

```
names
!
interface GigabitEthernet1/1
 nameif INSIDE
 security-level 100
 ip address 192.168.6.2 255.255.255.0
!
interface GigabitEthernet1/2
 nameif OUTSIDE
 security-level 0
 ip address 203.0.113.2 255.255.255.0
!
interface GigabitEthernet1/3
 no nameif
 no security-level
 no ip address
 shutdown
!
interface GigabitEthernet1/4
 no nameif
 no security-level
 no ip address
 shutdown
!
interface GigabitEthernet1/5
 no nameif
 no security-level
 no ip address
 shutdown
!
interface GigabitEthernet1/6
 no nameif
 no security-level
 no ip address
 shutdown
!
interface GigabitEthernet1/7
 no nameif
 no security-level
 no ip address
 shutdown
!
interface GigabitEthernet1/8
 no nameif
 no security-level
```

```
 no ip address
 shutdown
!
interface Management1/1
 management-only
 no nameif
 no security-level
 no ip address
 shutdown
!
object network IoT-OUT
 subnet 192.168.0.0 255.255.255.0
 nat (INSIDE,OUTSIDE) dynamic interface
!
route OUTSIDE 0.0.0.0 0.0.0.0 203.0.113.1 1
route INSIDE 192.168.0.0 255.255.255.0 192.168.6.1 1
route INSIDE 192.168.1.0 255.255.255.0 192.168.6.1 1
route INSIDE 192.168.2.0 255.255.255.0 192.168.6.1 1
route INSIDE 192.168.3.0 255.255.255.0 192.168.6.1 1
route INSIDE 192.168.4.0 255.255.255.0 192.168.6.1 1
route INSIDE 192.168.6.0 255.255.255.0 192.168.6.1 1
route INSIDE 172.16.1.0 255.255.255.0 192.168.6.1 1
route INSIDE 172.16.2.0 255.255.255.0 192.168.6.1 1
route INSIDE 172.16.3.0 255.255.255.0 192.168.6.1 1
route INSIDE 192.168.5.0 255.255.255.0 192.168.6.1 1
!
access-list OUTBOUND extended permit icmp object IoT-OUT any echo
access-list OUTBOUND extended permit icmp any object IoT-OUT echo-reply
access-list OUTBOUND extended deny tcp any any eq 1025
!
!
access-group OUTBOUND out interface INSIDE
!
!
class-map inspection_default
 match default-inspection-traffic
!
policy-map type inspect dns preset_dns_map
 parameters
  message-length maximum 512
policy-map global_policy
 class inspection_default
  inspect dns preset_dns_map
  inspect ftp
  inspect icmp
```

```
  inspect tftp
!
service-policy global_policy global
!
telnet timeout 5
ssh timeout 5
!
!
!
!
router ospf 1
 router-id 2.1.2.1
 log-adjacency-changes
 network 192.168.6.0 255.255.255.0 area 0
 network 203.0.113.0 255.255.255.0 area 0
!
```

Appendix H

APPENDIX

H.1 FIREWALL 2 CLI

```
: Saved
: Written by enable_15 at 04:39:02 UTC Mar 1 1993
: Call-home enabled from prompt by enable_15 at 04:39:02 UTC Mar 1 1993
:
ASA Version 9.6(1)
!
hostname FIREWALL2
names
!
interface GigabitEthernet1/1
 nameif OtherDepartment
 security-level 70
```

```
 ip address 192.168.0.2 255.255.255.0
!
interface GigabitEthernet1/2
 nameif INSIDE
 security-level 100
 ip address 192.168.7.2 255.255.255.0
!
interface GigabitEthernet1/3
 no nameif
 no security-level
 no ip address
 shutdown
!
interface GigabitEthernet1/4
 no nameif
 no security-level
 no ip address
 shutdown
!
interface GigabitEthernet1/5
 no nameif
 no security-level
 no ip address
 shutdown
!
interface GigabitEthernet1/6
 no nameif
 no security-level
 no ip address
 shutdown
!
interface GigabitEthernet1/7
 no nameif
 no security-level
 no ip address
 shutdown
!
interface GigabitEthernet1/8
 no nameif
 no security-level
 no ip address
 shutdown
!
interface Management1/1
 management-only
```

```
 no nameif
 no security-level
 no ip address
 shutdown
!
object network IoT-Network
 subnet 192.168.5.0 255.255.255.0
 nat (INSIDE,OtherDepartment) dynamic interface
object network IoT_Network
!
route INSIDE 192.168.7.0 255.255.255.0 198.168.7.1 1
route INSIDE 192.168.5.0 255.255.255.0 198.168.7.1 1
route OtherDepartment 192.168.0.0 255.255.255.0 192.168.0.1 1
route OtherDepartment 192.168.1.0 255.255.255.0 192.168.0.1 1
route OtherDepartment 192.168.2.0 255.255.255.0 192.168.0.1 1
route OtherDepartment 192.168.3.0 255.255.255.0 192.168.0.1 1
route OtherDepartment 192.168.4.0 255.255.255.0 192.168.0.1 1
route OtherDepartment 192.168.6.0 255.255.255.0 192.168.0.1 1
!
access-list OUTBOUND extended permit icmp object IoT-Network any echo
access-list OUTBOUND extended permit icmp any object IoT-Network echo-reply
access-list OUTBOUND extended deny tcp any any eq 1025
!
!
access-group OUTBOUND out interface INSIDE
!
!
class-map inspection_default
 match default-inspection-traffic
!
policy-map type inspect dns preset_dns_map
 parameters
  message-length maximum 512
policy-map global_policy
 class inspection_default
  inspect dns preset_dns_map
  inspect ftp
  inspect icmp
  inspect tftp
!
service-policy global_policy global
!
telnet timeout 5
ssh timeout 5
!
```

```
!
!
!
router ospf 1
 router-id 2.1.2.6
 log-adjacency-changes
 network 192.168.0.0 255.255.255.0 area 0
 network 192.168.7.0 255.255.255.0 area 0
!
```

Appendix I

APPENDIX

I.1 IDPS ROUTER CLI

```
!
version 15.4
service timestamps log datetime msec
no service timestamps debug datetime msec
service password-encryption
!
hostname IDPS-ROUTER
!
!
!
enable secret 5 $1$mERr$hbENBfUeQECKooMANN4/2/
!
!
!
!
!
!
ip cef
no ipv6 cef
!
```

```
!
!
username SDNnetworksecurity secret 5 $1$mERr$C37ombOqmsN4eydMeSm6r1
!
!
!
!
!
!
!
!
ip domain-name networksecurity.com
!
!
spanning-tree mode pvst
!
ip ips config location flash:ipsdirec retries 1
ip ips name iosips
ip ips signature-category
  category all
   retired true
  category ios_ips basic
   retired false
!
!
!
!
!
interface GigabitEthernet0/0/0
 ip address 192.168.7.1 255.255.255.0
 duplex auto
 speed auto
!
interface GigabitEthernet0/0/1
 ip address 192.168.5.1 255.255.255.0
 ip ips iosips out
 duplex auto
 speed auto
!
interface GigabitEthernet0/0/2
 no ip address
 duplex auto
 speed auto
 shutdown
!
```

```
interface Vlan1
 no ip address
 shutdown
!
router ospf 1
 router-id 2.1.2.7
 log-adjacency-changes
 network 192.168.7.0 0.0.0.255 area 0
 network 192.168.5.0 0.0.0.255 area 0
 network 192.168.0.0 0.0.0.255 area 0
!
ip classless
!
ip flow-export version 9
!
!
!
!
!
!
!
logging 192.168.5.100
line con 0
 exec-timeout 5 0
 password 7 08220D5D2A16254445
 logging synchronous
 login
!
line aux 0
!
line vty 0 4
 exec-timeout 5 0
 password 7 08220D5D2A16254445
 logging synchronous
 login
 transport input ssh
!
!
ntp server 192.168.5.100
!
end
```

Appendix J

APPENDIX

J.1 ROUTER1 CLI

```
!
version 15.4
service timestamps log datetime msec
no service timestamps debug datetime msec
service password-encryption
!
hostname ROUTER1
!
!
!
enable secret 5 $1$mERr$hbENBfUeQECKooMANN4/2/
!
!
!
!
!
!
no ip cef
no ipv6 cef
!
!
!
username SDNnetworksecurity secret 5 $1$mERr$C37ombOqmsN4eydMeSm6r1
!
!
!
!
!
!
```

```
!
!
ip ssh version 2
ip domain-name networksecurity.com
ip name-server 192.168.4.2
!
!
spanning-tree mode pvst
!
!
!
!
!
!
interface GigabitEthernet0/0/0
 description Connected to Ethernet network
 ip address 192.168.1.1 255.255.255.0
 duplex auto
 speed auto
!
interface GigabitEthernet0/0/1
 description Connected to Ethernet network
 ip address 192.168.2.1 255.255.255.0
 duplex auto
 speed auto
!
interface GigabitEthernet0/0/2
 no ip address
 duplex auto
 speed auto
 shutdown
!
interface Serial0/1/0
 description Connected to router Router2
 ip address 172.16.1.1 255.255.255.252
 clock rate 56000
!
interface Serial0/1/1
 description Connected to router Router3
 ip address 172.16.3.2 255.255.255.252
!
interface Vlan1
 no ip address
 shutdown
!
```

```
router ospf 1
 router-id 2.1.2.2
 log-adjacency-changes
 auto-cost reference-bandwidth 1000
 network 192.168.1.0 0.0.0.255 area 0
 network 192.168.2.0 0.0.0.255 area 0
 network 172.16.1.0 0.0.0.3 area 0
 network 172.16.3.0 0.0.0.3 area 0
!
router rip
!
ip classless
!
ip flow-export version 9
!
!
!
!
!
!
!
logging 192.168.4.2
logging 192.168.4.4
line con 0
 exec-timeout 5 0
 password 7 08220D5D2A16254445
 logging synchronous
 login
!
line aux 0
!
line vty 0 4
 exec-timeout 5 0
 password 7 08220D5D2A16254445
 logging synchronous
 login local
 transport input ssh
!
!
ntp server 192.168.4.4
!
end
```

Appendix K

APPENDIX

K.1 ROUTER2 CLI

```
!
version 15.4
service timestamps log datetime msec
no service timestamps debug datetime msec
service password-encryption
!
hostname ROUTER2
!
!
!
enable secret 5 $1$mERr$hbENBfUeQECKooMANN4/2/
!
!
!
!
!
!
no ip cef
no ipv6 cef
!
!
!
username SDNnetworksecurity secret 5 $1$mERr$C37ombOqmsN4eydMeSm6r1
!
!
!
!
!
!
```

```
!
!
ip ssh version 2
ip domain-name networksecurity.com
ip name-server 192.168.4.2
!
!
spanning-tree mode pvst
!
class-map match-any Business-Irrelevant
 match protocol icmp
!
policy-map PT_CONTROLLER_QUEUING_OUT
 class Business-Irrelevant
  bandwidth remaining percent 38
!
policy-map PT_CONTROLLER_MARKING_IN
 class Business-Irrelevant
  set ip dscp cs1
!
!
!
!
!
interface GigabitEthernet0/0/0
 description Connected to Ethernet network
 ip address 192.168.0.1 255.255.255.0
 service-policy input PT_CONTROLLER_MARKING_IN
 service-policy output PT_CONTROLLER_QUEUING_OUT
 duplex auto
 speed auto
!
interface GigabitEthernet0/0/1
 description Connected to Ethernet network
 ip address 192.168.6.1 255.255.255.0
 service-policy input PT_CONTROLLER_MARKING_IN
 service-policy output PT_CONTROLLER_QUEUING_OUT
 duplex auto
 speed auto
!
interface GigabitEthernet0/0/2
 no ip address
 service-policy input PT_CONTROLLER_MARKING_IN
 service-policy output PT_CONTROLLER_QUEUING_OUT
 duplex auto
```

```
 speed auto
 shutdown
!
interface Serial0/1/0
 description Connected to router Router3
 ip address 172.16.2.1 255.255.255.252
 service-policy input PT_CONTROLLER_MARKING_IN
 service-policy output PT_CONTROLLER_QUEUING_OUT
 clock rate 56000
!
interface Serial0/1/1
 description Connected to router Router1
 ip address 172.16.1.2 255.255.255.252
 service-policy input PT_CONTROLLER_MARKING_IN
 service-policy output PT_CONTROLLER_QUEUING_OUT
!
interface Vlan1
 no ip address
 shutdown
!
router ospf 1
 router-id 2.1.2.3
 log-adjacency-changes
 auto-cost reference-bandwidth 1000
 network 192.168.6.0 0.0.0.255 area 0
 network 172.16.1.0 0.0.0.3 area 0
 network 172.16.2.0 0.0.0.3 area 0
 network 192.168.0.0 0.0.0.255 area 0
!
router rip
!
ip classless
!
ip flow-export version 9
!
!
!
!
!
!
!
logging 192.168.4.4
line con 0
 exec-timeout 5 0
 password 7 08220D5D2A16254445
```

```
 logging synchronous
 login
!
line aux 0
!
line vty 0 4
 exec-timeout 5 0
 password 7 08220D5D2A16254445
 logging synchronous
 login local
 transport input ssh
!
!
ntp server 192.168.4.4
!
end
```

Appendix L

APPENDIX

L.1 ROUTER 3 CLI

```
!
version 15.4
service timestamps log datetime msec
no service timestamps debug datetime msec
service password-encryption
!
hostname ROUTER3
!
!
!
enable secret 5 $1$mERr$hbENBfUeQECKooMANN4/2/
!
```

```
!
!
!
!
!
no ip cef
no ipv6 cef
!
!
!
username SDNnetworksecurity secret 5 $1$mERr$C37ombOqmsN4eydMeSm6r1
!
!
!
!
!
!
!
!
ip ssh version 2
ip domain-name networksecurity.com
ip name-server 192.168.4.2
!
!
spanning-tree mode pvst
!
!
!
!
!
!
interface GigabitEthernet0/0/0
 description Connected to Ethernet network
 ip address 192.168.3.1 255.255.255.0
 duplex auto
 speed auto
!
interface GigabitEthernet0/0/1
 description Connected to Ethernet network
 ip address 192.168.4.1 255.255.255.0
 duplex auto
 speed auto
!
interface GigabitEthernet0/0/2
 no ip address
```

```
 duplex auto
 speed auto
 shutdown
!
interface Serial0/1/0
 description Connected to router Router1
 ip address 172.16.3.1 255.255.255.252
 clock rate 56000
!
interface Serial0/1/1
 description Connected to router Router2
 ip address 172.16.2.2 255.255.255.252
!
interface Vlan1
 no ip address
 shutdown
!
router ospf 1
 router-id 2.1.2.4
 log-adjacency-changes
 auto-cost reference-bandwidth 1000
 network 192.168.3.0 0.0.0.255 area 0
 network 192.168.4.0 0.0.0.255 area 0
 network 172.16.2.0 0.0.0.3 area 0
 network 172.16.3.0 0.0.0.3 area 0
!
ip classless
!
ip flow-export version 9
!
!
!
!
!
!
!
logging 192.168.4.2
logging 192.168.4.4
line con 0
 exec-timeout 5 0
 password 7 08220D5D2A16254445
 logging synchronous
 login
!
line aux 0
```