RASPBERRY HOUSE: AN INTRUSION DETECTION AND PREVENTION SYSTEM FOR INTERNET OF THINGS (IOT)

by

Wen Fei

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
March 2022

This thesis is dedicated to my parents, Dong and Huimin, who have always loved me unconditionally and has been a constant source of support and encouragement during my graduate study.

Table of Contents

List of Tables . . . . . **vi**

List of Figures . . . . . **viii**

Abstract . . . . . **x**

Acknowledgements . . . . . **xi**

Chapter 1 Introduction **1**

1.1 DoS Attacks on IoT 2

1.2 Strategies for Mitigating IoT DoS Attacks 3

1.3 Research Problem 4

1.4 Objectives 5

1.5 Thesis Outline 6

Chapter 2 Background **7**

2.1 IoT Architecture 7

2.2 Different Types of DoS Attacks 10
2.2.1 Deauthentication Attack 11
2.2.2 ICMP Flooding Attack 14
2.2.3 SYN Flooding Attack 15
2.2.4 Bash Fork Bomb 17
2.2.5 inode Problem 18

2.3 Raspberry Pi Gateway Components 20
2.3.1 Raspberry Pi 20
2.3.2 Raspberry House 21
2.3.3 MQTT 22
2.3.4 systemd 22
2.3.5 ATtiny85 23

2.4 Snort 25
2.4.1 Snort Rules 26
2.4.2 Snort inline mode 28

Chapter 3 Literature Review **31**

3.1 Comparative Study of Current Security Schemes for DoS Attacks . . 31

3.2 IoT Gateway Design 35

3.3 IDS for DoS/DDoS Attacks in IoT Security 36

3.4 IPS for DoS/DDoS Attacks in IoT Security 38

3.5 Research Gaps 39

3.6 Contributions 39

Chapter 4 Proposed Raspberry House Design **41**

4.1 Raspberry House Architectural Design 41

4.2 Raspberry House IDS and IPS Design 43
4.2.1 Raspberry House IDS Design 43
4.2.2 Raspberry House IPS Design 46

4.3 Raspberry House IDS and IPS Evaluation Design 48

4.4 Advantages of The Designed Architecture 48

Chapter 5 Experimental Implementation **53**

5.1 Raspberry House Architecture Implementation 53

5.2 Enable USB to TTL Connection on Attacker PC 57
5.2.1 Install prerequisites on Raspberry Pi 4 58
5.2.2 Enabling Connection on Raspberry Pi 4 58
5.2.3 Start The UART Connection 58

5.3 Raspberry House IDS and IPS Implementation 59
5.3.1 Data Link Layer IDS and IPS Implementation 66
5.3.2 Network Layer and Transport Layer IDS and IPS Implementation 74
5.3.3 System Security Level IDS and IPS Implementation 75
5.3.4 Evaluation Scenario Implementation 77

Chapter 6 Experimental Results and Evaluation **78**

Chapter 7 Conclusion and Future Work **88**

7.1 Conclusion 88

7.2 Discussion 88

7.3 Future Work 89

Bibliography **90**

List of Tables

2.1	Possible DoS Attacks in IoT	12
2.2	Sub-types of Management Frames	14
2.3	systemd Unit Types [1]	24
2.4	Snort Modes [2]	26
2.5	Snort Rule Actions [2, 3]	27
2.6	Snort Attack Classifications [4]	29
2.7	Snort Attack Classifications (cont'd.)	30
3.1	Comparison of Raspberry House with Other Related Research	32
3.2	Comparison of Raspberry House with Other Related Research (cont'd.)	33
4.1	Description of the Proposed Watchdog Timer (WDT) Schemes	48
4.2	Proposed Raspberry House's IDS and IPS Design	49
5.1	Pin Assignments Between GBKA and NodeMCU	57
5.2	Summary of the Systemd Services and Shell Scripts we used in thesis	60
5.3	Summary of the Systemd Services and Shell Scripts we used in thesis	61
5.4	Summary of the Systemd Services and Shell Scripts we used in thesis (cont'd.)	62
5.5	Summary of the Systemd Services and Shell Scripts we used in thesis (cont'd.)	63
5.6	Summary of the Systemd Services and Shell Scripts we used in thesis (cont'd.)	64
5.7	Summary of the Systemd Services and Shell Scripts we used in thesis (cont'd.)	65

5.8 Summary of the Systemd Services and Shell Scripts we used in thesis (cont'd.) 66

5.9 Summary of the Systemd Services and Shell Scripts we used in thesis (cont'd.) 67

5.10 Summary of the Systemd Services and Shell Scripts we used in thesis (cont'd.) 68

5.11 Summary of the Systemd Services and Shell Scripts we used in thesis (cont'd.) 69

5.12 Classify Services Based on MQTT Topic 70

5.13 The Functionality Provided By The Kotoriotoko [5] 73

5.14 Pin Assignments of the ATtiny85 and the Raspberry House . . 76

6.1 Raspberry House IPS Recommendation 86

List of Figures

2.1	IoT Five Layer Architecture [6]	8
2.2	Four Stages of IoT Architecture [7, 8]	9
2.3	802.11 Association Process	13
2.4	TCP Three-Way Handshake Process	16
2.5	Direct SYN Flood Attack	17
2.6	SYN Spoofed Attack	18
2.7	DDoS SYN Attack	19
2.8	Fork Bomb Attack	20
2.9	Raspberry Pi 3B+ Dimensions	21
2.10	Raspberry Pi OS	23
2.11	ATtiny85 (left) and Its Dimensions (right)	25
4.1	Raspberry House Architecture Design	42
4.2	External WDT Workflow	50
4.3	Raspberry House IDS and IPS Evaluation Design	51
5.1	Connection of Raspberry House	54
5.2	Connection Between Raspberry House and Universal PCB	55
5.3	Hardware Overview of GBKA [9]	56
5.4	GBKA Dimensions [9]	56
5.5	Headless Kali Linux UART Connection	59
5.6	UART Connection Result	59
5.7	Monitor Mode Setup Result	71
5.8	Monitor Mode Interface Chckeing	71
5.9	Content of enableAdapter.sh	72
5.10	Content of enableAdapter.service	72

5.11 Raspberry House Custom Snort Rules 74
5.12 Built in WDT Service Status 76

6.1 Deauthentication Attack Detection Results 79
6.2 Deauthentication Attack Alert Through Email 79
6.3 Deauthentication Attack Alert Through Twitter Message . . . 79
6.4 Normal Interfaces on Raspberry House 80
6.5 Raspberry House Interfaces Under Deauthentication Attack . 80
6.6 ICMP Flood DoS Attack Alert Through Email 81
6.7 ICMP Flood DoS Attack Alert Through Twitter Message . . . 81
6.8 SYN Flood DoS Attack Alert Through Email 81
6.9 SYN Flood DoS Attack Alert Through Twitter Message . . . 81
6.10 ICMP Flood DoS Attack IPS Drops Malicious Packets 82
6.11 SYN Flood DoS Attack IPS Drops Malicious Packets 82
6.12 WDTs Test Result of Bash Fork Bomb Attack 83
6.13 Gentle WDT Result 83
6.14 Delayed WDT Result 84
6.15 Gentle WDT Alert Through Email 84
6.16 Gentle WDT Alert Through Twitter Message 85
6.17 Delayed WDT Alert Through Email 85
6.18 Delayed WDT Alert Through Twitter Message 85

Abstract

It is estimated that the number of connected IoT (Internet of Things) devices and sensors will grow to around 125 billion by the end of this decade, up from an estimated 21 billion this year. IoT promises enormous benefits in several applications such as smart homes, smart cities, smart environment, agriculture, critical infrastructure control, and smart health. However, as the number of IoT devices increases and more information is shared among IoT devices, providing security becomes a top concern for researchers and developers. IoT devices have low power, and limited computing and storage capabilities, thus making them vulnerable to several attacks.

The objective of this thesis is to propose Raspberry House, a security gateway for detection and prevention of intrusions on IoT devices. Specifically, the gateway targets one of the major attacks on IoT devices, namely, Denial of Service (DoS), at the data link, network, transport layers and the system security level. The proposed gateway has been implemented using Raspberry Pi 3B+ and experimental analysis has been carried out against several DoS attacks, such as the deauthentication attack, SYN flood attack, ICMP flood attack, and the bash fork bomb attack. The results show that the Raspberry House can detect, alert, and prevent these DoS attacks in real time, and is particularly applicable to small IoT devices with resource constraints.

Acknowledgements

I would like to express my deepest and sincere gratitude to my supervisor, Prof. Srinivas Sampalli, for his guidance, support, encouragement and patience during my study and for providing all the necessary opportunities to make this work possible.

I am extremely grateful to Prof. Hiroyuki Ohno (Kanazawa University, Japan) for his support and guidance in the implementation of this research. His expertise in IoT devices and attention to details have greatly improved this research.

I am very appreciative of Darshana, Richard, Mahdieh, and Nupur's continuous support and encouragement.

Last but not least, I want to thank the members of MYTech Lab for their patience and infinite kindness, and thank them for their consistent presence whenever I needed them.

Chapter 1

Introduction

According to the latest report from Juniper Networks Research, the total number of connected IoT sensors and devices is set to exceed 50 billion by the end of 2022, up from an estimated 21 billion in 2018 [10]. This number is expected to grow to 125 billion by 2030, which means that each IoT user is expected to have around 15 connected devices [11].

IoT devices use embedded systems such as processors, sensors, and actuators to collect, send, and process data obtained from the environment [12, 13]. These devices share the sensor data they collect by connecting to IoT gateways or other edge devices, and this data will be analyzed locally or sent to the cloud for analysis [13]. IoT devices can also communicate with other IoT devices, share valuable information with applications, and take actions based on the acquired information [12, 14]. In addition, these devices can automate different tasks, which enables physical objects to take action without any human intervention [14].

IoT brings enormous benefits to several applications. For example, smart agriculture can use the rainfall, humidity, temperature, and soil oxygen content collected by sensors to monitor soil quality and make decisions based on the results to achieve efficient use of water, electricity and other resources [7, 15]. Smart transportation collects and transmits data through embedded sensors, actuators and other IoT devices, providing users with real-time traffic conditions and helping them plan their journey path to avoid traffic jams and hazardous conditions [7, 16]. Smart homes allow users to control IoT devices through the network remotely. For instance, they can monitor room temperature and the power consumption of smart appliances through applications, thereby reducing resource waste [12, 7]. In the area of healthcare, implantable and wearable smart health devices collect the patient's heart rate, blood pressure and other health indicators, and send this data to an application that healthcare professionals can check, enabling healthcare professionals to improve existing services based

on the real-time data obtained, helping them to make better decisions [7]. With the popularity of the IoT in the market, organizations and companies are increasingly using IoT to improve operational efficiency. They use automated tasks to provide customers with better services, improve decision-making and increase the business value [7, 17]. IoT also reduces labour costs and generates more revenue. In addition, since technicians can access information on any IoT device anytime, anywhere, they can continuously improve communication between connected IoT devices.

However, as the number of IoT devices increases and more information is shared among such devices, they become more vulnerable to cyber-attacks. Attackers can exploit security vulnerabilities in IoT infrastructures to perform complex cyber attacks. These cyber-attacks include Denial of Service (DoS) attacks, man-in-the-middle (MITM) attacks, replay attacks, routing attacks, data breaches, and security and privacy threats. The limited resources of small IoT devices make it difficult to build security protection measures in small IoT devices, making them more vulnerable to various attacks. Attackers can exploit vulnerabilities in IoT systems to manipulate the system's data, thus making it unusable [16, 18, 19].

IoT has low power and limited computing and storage capabilities, making it easy for attackers to use these flaws to access network data such as homes and businesses [17]. In addition, due to profit-driven choices and lack of relevant legislation, many manufacturers do not provide users with patches and updates after the product is produced, leading to IoT device security risks. For example, since IoT devices such as routers cannot be updated regularly, they are vulnerable to botnets. Moreover, due to the rapid development of smart healthcare, attackers on medical IoT devices can lead to leakage of patient medical data or even loss of life.

1.1 DoS Attacks on IoT

Denial of Service (DoS) is one of the most common IoT attacks. In 2016, the Mirai attacks attracted widespread public attention [20]. Attackers used poorly secured IoT devices to infiltrate the Domain Name Server (DNS) provider Dyn and attack many websites [20], which is one of the largest distributed denial of service (DDoS) attacks, which infected around 2.5 million internet-connected devices and caused Dyn to lose nearly 8% of its customers [14, 17].

A DoS attack [7, 21] typically floods the target host or network by sending multiple requests until the target fails to respond or crashes due to traffic overload, preventing legitimate users from accessing expected services or resources. Furthermore, it can slow down or disable services. According to Vaadata's report [22], DoS attacks are one of the most frequently occurring attacks in IoT. In addition, according to Tsiatsis et al. [23], DoS attacks are the most common and easiest attacks on IoT systems and are defined as attacks that undermine the ability of the network or system to perform expected tasks. For example, a DoS attack can undermine the wireless network of enterprise IoT devices, causing business disruption, lost revenue from system downtime, or even damage to a brand's reputation. DoS attacks can also continue for weeks or even months, causing IoT devices to fail to run continuously.

1.2 Strategies for Mitigating IoT DoS Attacks

Previous research has focused on a number of strategies for DoS attack detection and prevention on IoT devices.

Kajwadkar and Jain [24] propose a novel solution to detect DoS attacks against restricted devices. Detection occurs in the early stages of the border router node, ensuring that no network devices in any IoT network are harmed. The detection method includes two stages, the primary stage and the secondary stage. In the primary stage, the source IP and packet size are checked, and the algorithm decides whether the source is a confirmed threat or suspicious. The second stage verifies the legitimacy of the suspicious input.

Mubarakali et al. [25] propose a machine learning SDN model based on the SVM algorithm. The controller in the proposed model is responsible for collecting the flow table status data on the network traffic switches and then uses the support vector machine (SVM) algorithm to extract twice-characteristic values related to DDoS attacks for traffic evaluation and DDoS attack detection. The results show that the proposed model can effectively detect DDoS attacks with a low false-positive rate.

Anirudh et al. [26] propose a honeypot model to mitigate DoS attacks launched on IoT devices. In the proposed model, the honeypot is used in the online server as a decoy for the main server so that DoS attacks are forwarded to this decoy protecting the target server to mitigate the attacks to the intended target server. The protection

is achieved by tracking attackers and tracing their activities to study further and analyze them to prevent future attacks.

Chifor et al. [27] propose a lightweight security mechanism for addressing DoS attacks in IoT publish-subscribe applications. The authors protect the message brokers against quality-of-service (QoS) based DoS attacks by designing an extensible security framework. Although the proposed framework helps to orchestrate IoT network applications from a security perspective, it suffers from the disadvantage of dealing with single authority management for trusted devices.

Mamun [28] builds an example of real-time integration of IoT sensors and the IOTA Tangle to improve the security of IoT infrastructure while testing the efficiency of the network. The results show that although the IOTA Tangle has not yet been deployed on a large scale to IoT devices and is still in an immature development stage, it has the potential to create a new dimension to secure IoT devices.

Among the above studies, most of them focus on the detection methods of DoS attacks against IoT devices. Although some of these studies provide the mitigation strategy, it can only be run after extensive simulation testing since it requires collecting more elements such as protocols and the packet size.

1.3 Research Problem

The research problems are as follows.

- Since most DoS attacks occur at the network layer and transport layer, many studies only focus on DoS attacks at these two layers. However, with the popularity of wireless local area networks (WLANs), many IoT devices are connected wirelessly and transmit messages. The rapid growth of wireless network deployments has attracted attackers to target them [29, 30]. The data link layer in wireless networks is more vulnerable to DoS attacks such as deauthentication attacks due to unprotected management frames [30]. Therefore, researchers also need to focus on DoS attacks at the data link layers of the TCP/IP model.

- System security level DoS attacks such as bash fork bomb can make the system overloaded and unable to respond to any input, so researchers also need to

consider Intrusion Detection System (IDS) and Intrusion Prevention System (IPS) for system security level DoS attacks.

- Since small IoT devices such as Microcontroller Units (MCUs) are inexpensive and easy to use, they are the first choice for labs and some enterprises' research institutions. However, small IoT devices are inherently resource-constrained, i.e., they have limited processing speed, storage capacity, and communication bandwidth [31]. Therefore, it is not easy to build and run the IDS and IPS.

- Furthermore, the solutions proposed by most of the research are limited to detecting DoS attacks in IoT without providing prevention solutions.

- As pointed out by Zachariah et al., Castellanos et al. and Gloria et al. [32, 33, 34], IoT-enabled IDS and IPS devices proposed in existing research work are expensive and inconvenient to carry.

The aforementioned research gaps form the motivation for this thesis.

1.4 Objectives

IoT devices have low power and limited computing and storage capabilities, thus making them vulnerable to several attacks. This thesis aims to design, implement, and test a gateway called Raspberry House, a security gateway for detecting and preventing intrusions on IoT devices. Specifically, the gateway targets one of the major attacks on IoT devices, namely, Denial of Service (DoS), at the data link, network, transport layers and the system security level. The proposed gateway has been implemented using Raspberry Pi 3B+, and experimental analysis has been carried out against several DoS attacks, such as the deauthentication attack, SYN flood attack, ICMP flood attack, and the bash fork bomb attack. In addition, the proposed IoT gateway is based on shell scripts, and in order to enable it to run detection and prevention mechanisms autonomously, our research also considers the use of systemd services. The results show that the Raspberry House can detect, alert, and prevent these DoS attacks in real-time, particularly applicable to small IoT devices with resource constraints.

1.5 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 provides the background work that helps to understand the proposed approach. Chapter 3 introduces the literature survey involved in the research and the research gaps. Chapter 4 discusses the design of the Raspberry House framework, Raspberry House IDS and IPS design, and the evaluation design of the Raspberry House. Chapter 5 and Chapter 6 describe the experimental implementation and evaluate the performance of the proposed approach. Chapter 7 summarizes the contributions of this thesis and discusses future work.

Chapter 2

Background

This chapter provides the background in the following areas that are necessary for the rest of the thesis:

- IoT Architecture
- DoS Attack Types
- Raspberry Pi Gateway Components
- Snort Security Tools

2.1 IoT Architecture

The IoT network needs to integrate all resources, hardware, software, and systems into a framework to form an integrated, reliable, and cost-effective solution [7, 16, 8]. Therefore, each IoT architecture needs to be developed according to its function and implementation in different domains, i.e., there is no single, standard-defined IoT architecture [8, 35]. Nevertheless, the foundation of each IoT architecture and its general data processing flow is roughly the same [8]. [7, 8, 36, 6] describes the typical IoT layered protocol stack and architecture. Iqbal et al. [8] outline the characteristics of IoT security and propose a general IoT architecture and IoT protocol stack. Swamy and Kota [7] summarize the architecture of IoT and its current status and analyze the status of the communication standards and application layer protocols used in IoT. Bouaouad et al. [36] analyze different IoT architectures and their layers in the cloud environment and determine the key layers to define a complete and detailed architecture. Al-Fuqaha et al. [6] discuss the overall architecture of IoT and its security issues. The three-layer and five-layer architectures are the most prominently used in these studies. Figure 2.1 shows the five-layer architecture. The three-layer

architecture is a subset of the five layers. A brief description of each layer is given below.

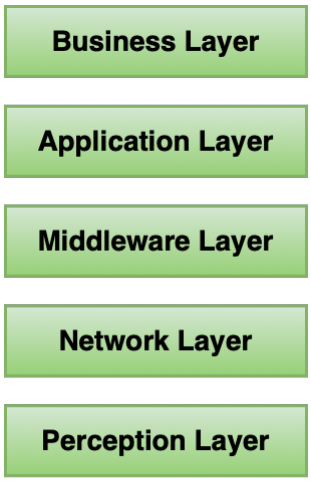

Figure 2.1: IoT Five Layer Architecture [6]

- Perception Layer

  The perception layer, also called the sensor layer, is the physical layer of the IoT architecture [6]. It is responsible for using sensors and embedded systems to identify objects and collect data from them [6].

- Network Layer

  The network layer carries and transmits the data collected by the sensors in a wired or wireless manner [7]. It is also responsible for connecting various smart objects, network devices and servers.

- Middleware Layer

  The middleware layer or service management layer has features such as data storage, computation, processing, and analysis [6].

- Application Layer

  The application layer manages all application processes based on the information obtained from the middleware layer. This layer is the interface between IoT devices and networks, interacting with users [6].

- Business Layer

  The business layer manages the entire IoT system, which involves constructing flowcharts, graphs, analyzing results, and improving IoT devices [6].

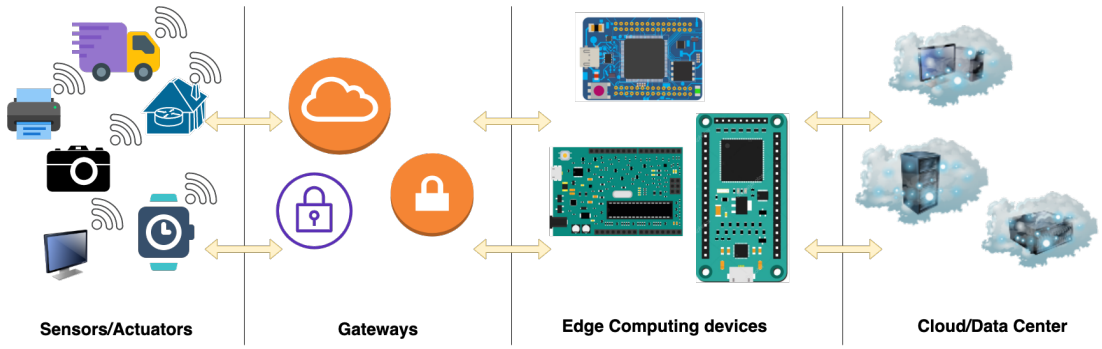

Figure 2.2: Four Stages of IoT Architecture [7, 8]

Figure 2.2 shows the four stages of IoT Architecture. As shown in Figure 2.2, data is the basis of every IoT system and can be provided by connected IoT devices [37]. An important feature of the sensors is the ability to convert information obtained from the outside world into data for analysis [38, 39, 40]. The actuator can transform the data generated by IoT devices into physical actions by cooperating with sensors [40, 41].

The gateway and data acquisition system (DAS) provides the necessary connection nodes to connect the remaining layers. Furthermore, the DAS collects raw data from sensors and converts it from an analog format to a digital format [42]. Then the DAS aggregates and formats the data and sends the data to the following processing stage through the gateway [42, 43].

Before entering the data center or cloud, digitized and aggregated IoT data needs to be processed to reduce its size further [44]. As part of preprocessing, edge systems can provide faster response time and greater flexibility in the processing and analysis of the IoT data [44, 45, 46].

The primary process of the final stage takes place in the data center or cloud. Data centers or cloud-based systems are designed to use powerful data analysis engines and machine learning mechanisms to store, process, and analyze large amounts of data to gain deeper insights [45, 47, 48].

2.2 Different Types of DoS Attacks

We introduce different types of DoS attacks according to each layer of the TCP/IP model to provide a clear understanding of DoS attacks.

DoS attacks can cause physical media to be disabled or manipulated maliciously at the physical layer, making it unavailable to users [49]. For example, the use of low-power micro-jammers to transmit wireless jamming signals to disrupt wireless communications maliciously, pull patch cords, and brute force or shutdown network devices are all DoS attacks at the physical layer [50]. Standard physical layer DoS attacks include jamming, node destruction and interference [51].

The data link layer ensures that data is efficiently transmitted to the physical layer. The primary communication protocol of this layer is the IEEE802 standards, such as IEEE802.11 (WiFi) and IEEE802.3(Ethernet) [49]. Therefore, DoS attacks at this layer are usually aimed at data frame detection, media access control, data stream multiplexing and error control [50]. Attacks at the data link layer include Content Address Table (CAM) exhaustion, Address Resolution Protocol (ARP) spoofing, MAC address spoofing, deauthentication attack, disassociation attack, etc.

Network layer DoS attacks are the injection of more traffic into the victim network than it can handle, which causes the victim network to start responding slowly or drop some packets [49]. However, the loss of packets can lead to an overflow of retransmission requests and extra traffic that can make the network unavailable to legitimate users [49, 50]. Network layer DoS attacks include ICMP flooding, Sybil attack, hello flooding, wormhole attack, smurf attack, etc.

The transport layer is susceptible to DoS attacks in the form of flooding by generating and transmitting large amounts of traffic to prevent the network's availability of services or resources for legitimate users [50]. In addition, a large amount of traffic can quickly consume device resources, especially for small IoT devices that are highly constrained, resulting in a decrease in the overall utility of the system [23]. DoS attacks at the transport layer include the abuse of TCP and UDP protocols to flood resources in the network, such as SYN floods and UDP floods [50].

Since most protocols in the application layer are built on a client-server model, where the server helps legitimate users to achieve a specific service and the client is a procedure of requesting services from the server [50]. Therefore, DoS attacks at

the application layer can be accomplished by sending a large number of legitimate requests to the server until the server is swamped. This attack can make a server very busy without breaking it, making it inaccessible to legitimate users. DoS attacks at the transport layer include HTTP GET flooding, HTTP POST flooding, PUSH flooding, HEAD flooding, SMTP Mail Flooding, etc [49].

In addition to DoS attacks against the TCP/IP model, DoS attacks at the system security level also deserve the attention of researchers. DoS attacks at the system security level mainly refer to the failure of the system to run properly by overloading shared resources or services. For example, a DoS attack at the system security level can generate multiple processes in parallel that fill up the disk so that the system can no longer support any processes or cannot execute any instructions, thereby turning down the entire system. System security level DoS attacks include bash fork bomb, inode problems, etc.

Table 2.1 shows the possible attacks based on the above classification criteria.

This thesis focus on designing IDS and IPS for the proposed gateway to effectively defend against DoS attacks at the data link layer, network layer and transport layer in the TCP/IP model. In addition, we also consider the system security level IDS and IPS for the proposed gateway, so it can also effectively prevent system-level DoS attacks. We now highlight four major attacks that are central to this thesis.

2.2.1 Deauthentication Attack

An Access Point (AP) is the bridge that connects traffic between client devices and other devices on the network. In order to be able to connect to an access point (AP), the client must have established a connected state before it can start exchanging data messages [29]. The following are the three 802.11 connection states [52]:

- Unauthenticated and Unassociated
- Authenticated but Unassociated
- Authenticated and Associated

The client and AP will exchange a series of 802.11 management frames for entering the authenticated and associated state. Figure 2.3 shows the 802.11 association process.

Table 2.1: Possible DoS Attacks in IoT

Classification Criteria		**Attacks**
TCP/IP Model	Application Layer	HTTP GET flooding HTTP POST flooding PUSH flooding HEAD flooding SMTP Mail Flooding
	Transport Layer	SYN flood attack UDP flood attack
	Network Layer	ICMP flooding Sybil attack Hello flooding Wormhole attack Smurf attack
	Data Link Layer	CAM exhaustion ARP spoofing MAC address spoofing Deauthentication attack Disassociation attack
	Physical Layer	Jamming Node destruction Interference
System Security Level		Bash fork bomb Inode problem

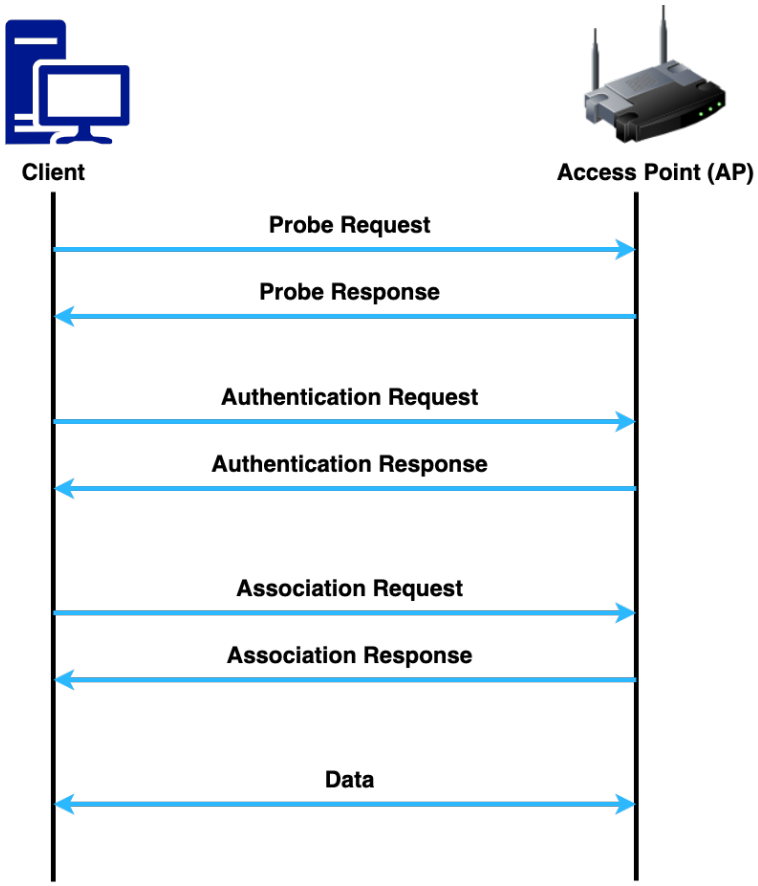

Figure 2.3: 802.11 Association Process

First, the access point periodically sends a beacon frame to announce its presence. When a client wants to join a network, it sends a probe request to discover 802.11 networks in its proximity. The AP receives the probe request and checks whether the client has at least one commonly supported data rate. If they have a compatible data rate, AP will send a probe response advertising the SSID, supported data rates, encryption types, and other 802.11 capabilities of the AP [29, 30, 53].

Then, the client and AP authenticate by exchanging messages. The client selects a compatible network from the probe responses it receives, and once a compatible network is found, the client will attempt low-level 802.11 authentication with the compatible AP [53]. The AP receives the authentication frame and authenticates any client attempting to join the network. After the client is authenticated, the client and AP are authenticated but not yet associated [29].

Next, the client confirms the AP to associate with, and it will send an association

Table 2.2: Sub-types of Management Frames

Type Value	Type Description	Subtype Value	Subtype Description
00	Management Frame	0000	Association Request
		0001	Association Response
		0010	Re-association Request
		0011	Re-association Response
		0100	Probe Request
		0101	Probe Response
		1000	Beacon
		1001	Announcement Traffic Indication Massage (ATIM)
		1010	Disassociation
		1011	Authentication
		1100	Deauthentication
		1111	Reserved

request to that AP. After receiving the association request, the AP will check whether it has the capabilities to respond [53]. After the confirmation, it will create an association ID for the client and respond with an association response and a success message that grants the client access [29, 53].

Finally, the client can successfully associate with the AP and start data transfer. It is worth noting that if the wireless network requires WPA/WPA2 or 802.1X authentication, the client will not be able to send data until dynamic keying and authentication have occurred after the 802.11 association is complete [29].

When clients wish to disconnect from the AP, they need to send a deauthentication frame to the AP. However, according to the 802.11 networking standard, deauthentication or disassociation frames are unencrypted and do not require authentication. Therefore, an attacker can use this to easily spoof a client or AP's MAC address to generate deauthentication packets on behalf of the client or AP [29, 53]. Deauthentication frames belong to the category of management frames, and Table 2.2 shows all 12 management frame subtypes identified by the 802.11 standards [30, 53].

2.2.2 ICMP Flooding Attack

Internet Control Message Protocol (ICMP) is a network layer protocol that network devices use to communicate [54, 55]. An ICMP flood DoS attack, also known as a

Ping flood attack, is a typical DoS attack in which an attacker attempts to flood a target device by using ICMP echo-request packets [13]. ICMP echo-request and echo-reply messages are used to ping network devices to diagnose the device's health and connectivity and the connection between the sender and the device [56, 55].

In this type of attack, the attacker floods the target with request packets, and the network is forced to respond with the same number of reply packets. As a result, both the incoming and outgoing channels of the network are overwhelmed, causing normal network activity to be interrupted, the target device inaccessible to normal traffic, and significant bandwidth consumed [56, 55]. As a Ping Flood attack floods the target device's network connection with fake traffic, legitimate requests cannot get through, resulting in a DoS attack on legitimate users [56].

2.2.3 SYN Flooding Attack

TCP SYN flood (aka SYN flood) is a DoS attack designed to exploit the TCP three-way handshake to consume resources on a target server and make it slow or unresponsive to legitimate traffic [57, 58]. It can target any system connected to the Internet and provides Transmission Control Protocol (TCP) services such as web servers, email servers, etc [57].

Figure 2.4 shows the process of establishing a standard TCP "three-way handshake" between client and server [57].

- First, the client requests to initiate a TCP connection by sending an SYN (Synchronization) message to the server.
- The server receives the message and responds to the client by sending an SYN-ACK (Synchronization Acknowledgment) message to the client.
- Finally, the client responds with an ACK (Acknowledgement) message and establishes the connection. After completing this packet sending and receiving series, the TCP connection can send and receive data.

The SYN flood attack is a TCP state exhaustion attack that attempts to consume the connection state tables in many infrastructure components, such as firewalls. In an SYN flood attack, the attacker uses a fake IP to act as a client, sending repeated

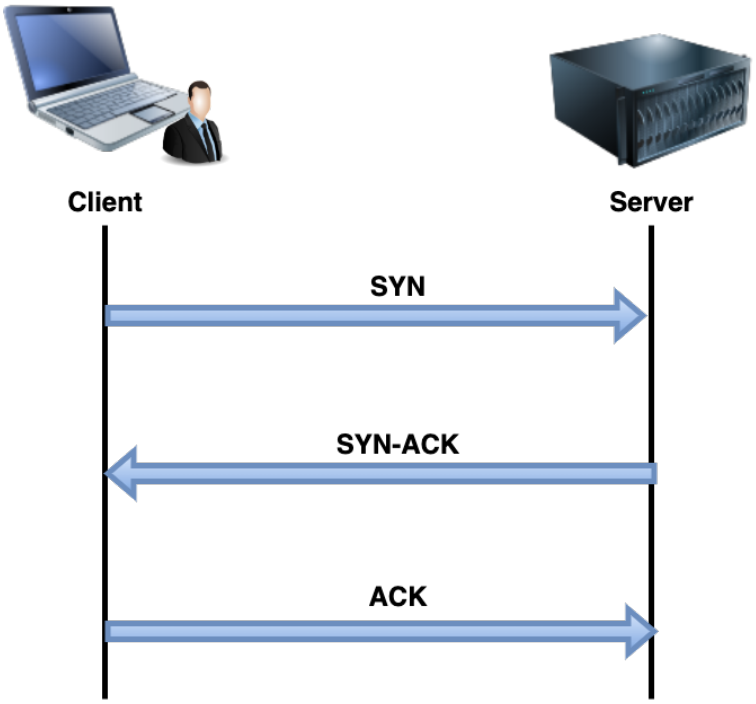

Figure 2.4: TCP Three-Way Handshake Process

TCP SYN connection requests at a higher rate than the target server can process. The following lists three common SYN flood attacks [57, 58].

- **Direct SYN Flood Attack:** As shown in Figure 2.5, The attacker sends multiple SYN requests to the target server using its IP address. The target server responds to the SYN request with the SYN-ACK packet. However, instead of sending the expected ACK response, the attacker continues to send new SYN requests, which causes the target server to keep waiting for an ACK of its SYN-ACK packet, leaving more and more links half-open [58]. Eventually, as the server's connection overflow table fills up, the target server becomes inoperable or crashes and rejects legitimate client requests.

- **SYN Spoofed Attack:** It is easy to detect in a direct SYN flood attack since the attacker uses his/her IP address. To avoid this, attackers send SYN packets using the spoofed IP address. As shown in Figure 2.6, after the server receives the spoofed SYN packets, it responds with SYN-ACK packets and waits for ACK response. However, the attacker will not send the ACK to the target server. In addition, to ensure that the spoofed IP address never responds to the

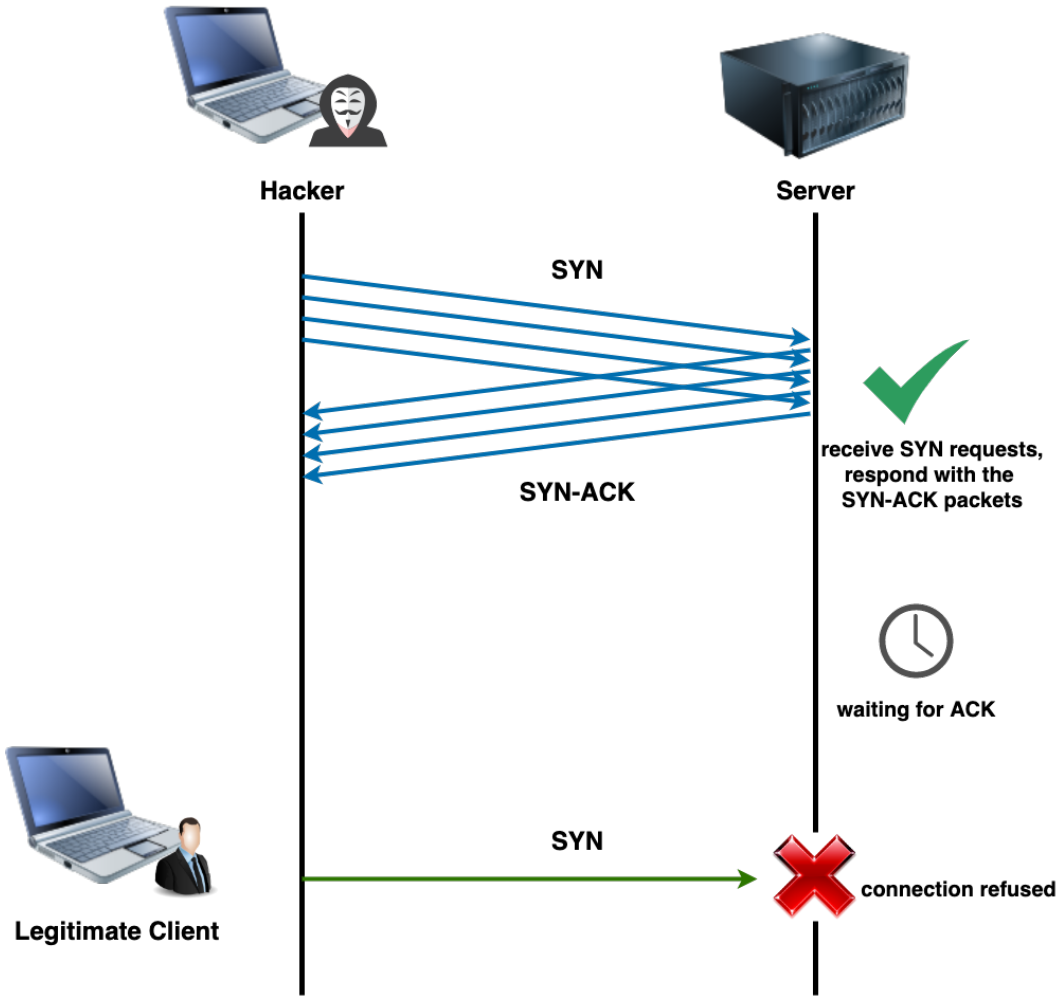

Figure 2.5: Direct SYN Flood Attack

SYN-ACK sent by the target server, the attacker uses the unused IP address.

- **DDoS SYN attack:** As shown in Figure 2.7, in the DDoS SYN attack, the attacker sends SYN packets to the target server by controlling a botnet consisting of multiple hijacked computers, making the target server inaccessible to legitimate users.

2.2.4 Bash Fork Bomb

The fork bomb is a DoS attack designed to consume the system resources. Fork bombs can be written in different programming languages, such as Python. In this thesis, we use the bash fork bomb. Figure 2.8 shows the process of the fork bomb attack,

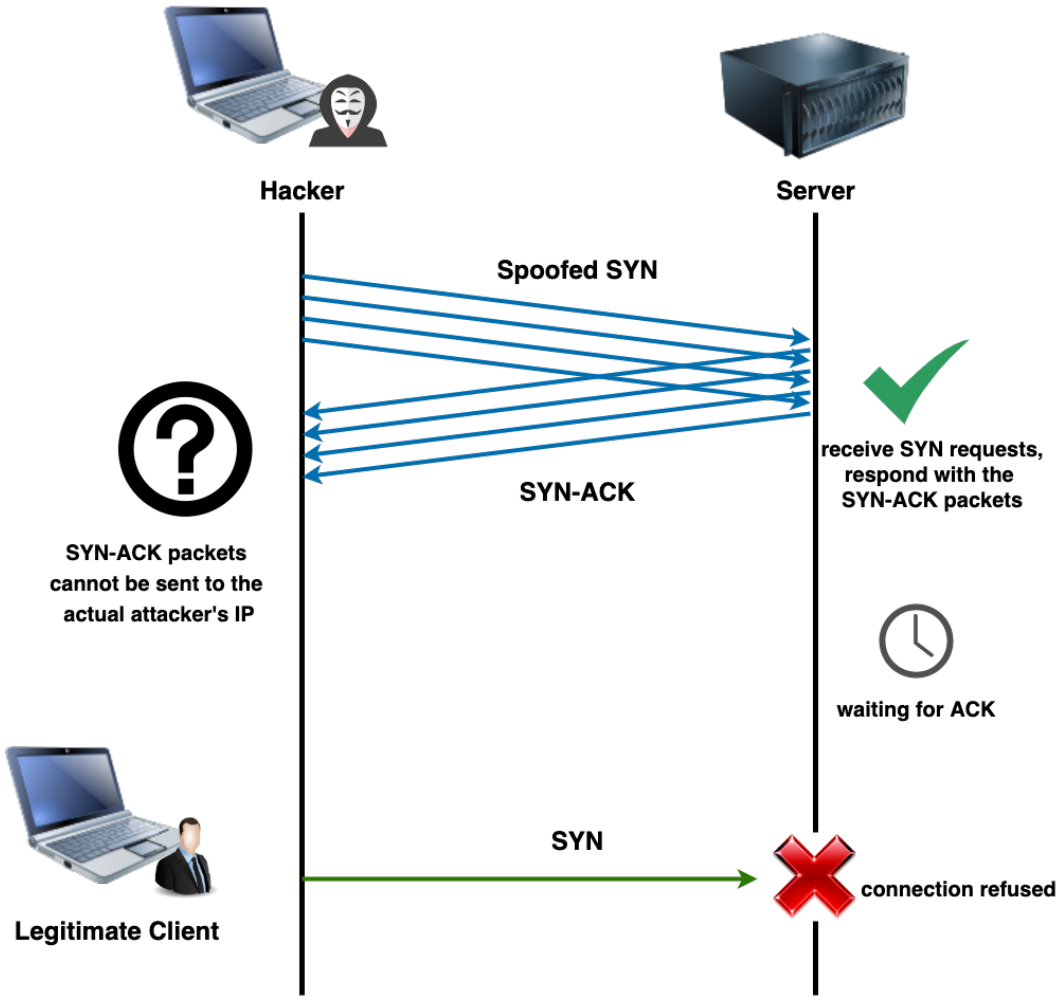

Figure 2.6: SYN Spoofed Attack

where the attacker first creates a self-replicating child process through malicious code. The processes recursively fill the entire available system resources, which causes the system to slow down or crash due to resource starvation [59, 60].

2.2.5 inode Problem

Linux file systems use inodes to store information about files, directories, and devices. Inodes are the basis of the Linux file system. Inodes manage metadata about files and are an important part of the inner workings of Linux. All files in any Linux directory have a filename and an inode number, and users can retrieve the file's metadata by referencing the inode number. Metadata includes file type, permissions, owner ID, group ID, file size, last access time, last modification time, soft/hard links, and

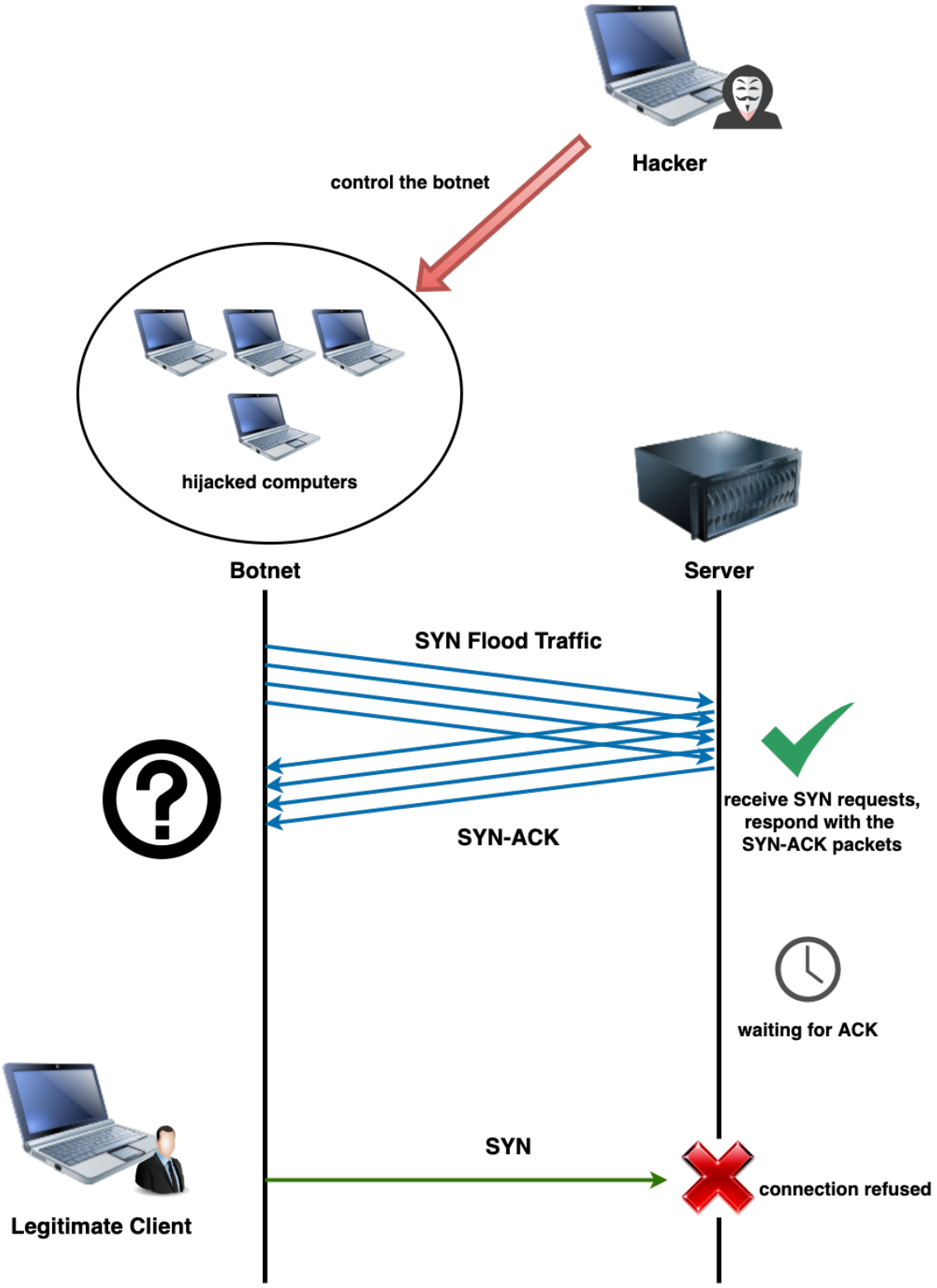

Figure 2.7: DDoS SYN Attack

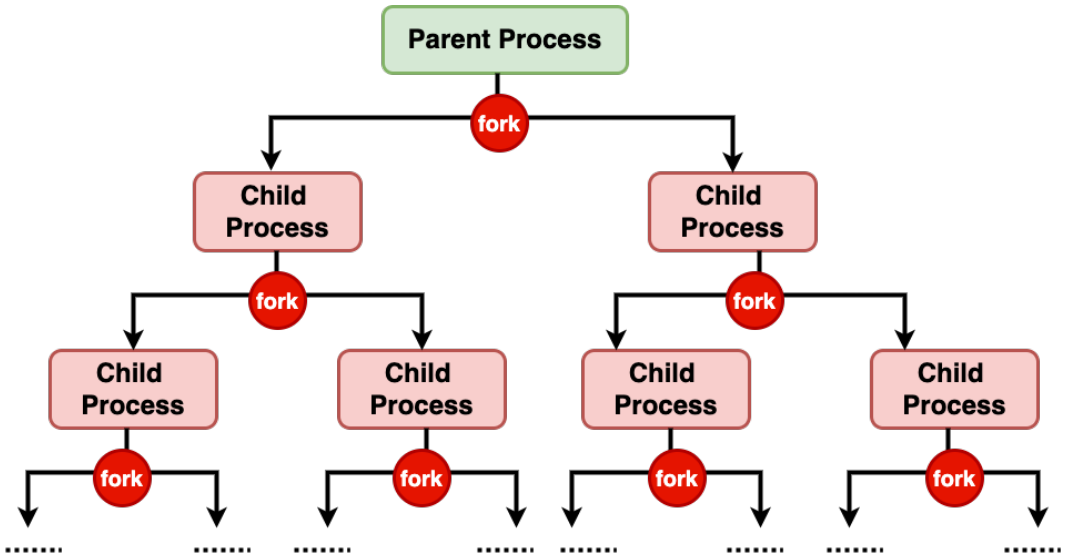

Figure 2.8: Fork Bomb Attack

access control lists. Therefore, an attacker can consume all the free inodes on the disk by creating a large number of temporary empty folders. Most systems ignore this seemingly normal behaviour of creating empty folders. Since this behaviour is difficult to detect, an attacker can create temporary empty folders until all free inodes on the system disk are consumed. Thus, new files cannot be created. If the supply of available inodes is exhausted, the system cannot allocate new files even if disk space is available.

2.3 Raspberry Pi Gateway Components

2.3.1 Raspberry Pi

As shown in Figure 2.9, the Raspberry Pi is a low-cost, credit-card-sized single-board computer designed to teach programming skills, build hardware projects, perform home automation, and explore industrial applications of computer technology [61]. According to Amazon, the Raspberry Pi 3B+ used in this thesis only costs $79 CA [62]. It is based on the Linux operating system and provides general-purpose input and output (GPIO) pins that allow users to control electronic components for physical computing and exploration of the IoT. Linux is an open-source operating system (OS) that connects a computer's hardware and software programs.

Figure 2.9: Raspberry Pi 3B+ Dimensions

We choose to use Raspberry Pi as the gateway because of its simplicity and security. Compared to Windows, whose source code is inaccessible to users, Linux is based on a multi-user architecture. Therefore, it is more stable than single-user OS such as Windows. Since Linux is community-driven and regularly monitored by developers worldwide, any new problems that arise can be resolved within a short period of time, and necessary patches can be provided at any time.

2.3.2 Raspberry House

The latest version of Raspberry Pi already has onboard wireless capabilities, which can act as many different devices. In our previous work [63], we have built Raspberry House as an IoT gateway using a Raspberry Pi 3B+. Raspberry House is an IoT security framework, and it can generate a private network and assign private IP addresses to small IoT devices so that the devices will not be exposed to outside networks. We have also implemented the wireless firmware update of the IoT devices

inside the gateway. Therefore, all the IoT devices inside the gateway can be updated synchronously without connecting them to the USB interface, which saves time. Furthermore, we have considered that in practical scenarios, each organization may invest in a large number of IoT devices to use, and these IoT devices may be distributed all over the world. Therefore, we need to consider the communication between IoT devices. In order to ensure that the communication is secure, we have tested TLS encryption technology, SSH port forwarding and cloud services, analyzed the security of IoT device communication and given suggestions for different environments.

2.3.3 MQTT

The publish/subscribe-based Message Queuing Telemetry Transport (MQTT) protocol is popular in IoT. MQTT, Message Queuing Telemetry Transport, is a lightweight IoT messaging protocol used to collect the measurement data from remote sensors and send it to the server via TCP [16, 64]. It uses a publish/subscribe (pub/sub) technology and allows simple data flow between different devices. The pub/sub model is based on an MQTT broker. Clients can publish a message on a topic or subscribe to a particular topic through the broker. Furthermore, topics represent the destination address for a message sent by the MQTT broker. The client who subscribes to a topic will receive all the published messages [65].

It also supports low-bandwidth and high-latency networks to minimize network bandwidth and equipment resources [16]. Although MQTT supports lightweight communication, it also has some limitations. Since it only supports TCP, which is limited by the network, packet loss frequently occurs [12].

2.3.4 systemd

When a Linux system boots up, systemd provides a standard procedure for controlling programs, including starting daemons on demand, mount and automatic mount point maintenance, the network stack, cron-style job scheduler, and process tracking using Linux control groups, etc [1]. Many Linux distributions use systemd to manage system settings and services. Raspberry Pi OS from the Jessi versions supports systemd. As shown in Figure 2.10, we use the latest version (i.e. Buster) of Raspberry Pi OS in this thesis, which supports systemd.

```
pi@wenpipi:~ $ cat /etc/os-release
PRETTY_NAME="Raspbian GNU/Linux 10 (buster)"
NAME="Raspbian GNU/Linux"
VERSION_ID="10"
VERSION="10 (buster)"
VERSION_CODENAME=buster
ID=raspbian
ID_LIKE=debian
HOME_URL="http://www.raspbian.org/"
SUPPORT_URL="http://www.raspbian.org/RaspbianForums"
BUG_REPORT_URL="http://www.raspbian.org/RaspbianBugs"
```

Figure 2.10: Raspberry Pi OS

SysVinit is an init system that predates systemd and uses a simplified method to start services. SysVinit leaves a lot to be desired [1]. For example, since shell scripts in SysVinit are executed line by line, system services are not executed in parallel, resulting in slow startup times. While systemd is backward compatible with SysVinit scripts, systemd is designed to replace these older ways of getting Linux systems running. systemd is able to start services in parallel and provides dependency-based service control logic to reduce startup time. In addition, systemd provides a tool called systemctl designed to interact with processes controlled by systemd. For example, systemctl can check the status of units and targets, as well as start and stop the services.

systemd introduces the concept of systemd units, which organize tasks into components called units, and groups of units into targets that can be used to create dependencies on other system services and resources [1]. Table 2.3 provides the complete systemd unit types [1].

2.3.5 ATtiny85

The ATtiny85 microcontroller is a RISC architecture-based AVR microcontroller manufactured by Microchip. The ATtiny85 is a low-power small microcontroller with 2096-byte flash memory and 512-byte EEPROM memory [66, 67]. In addition, it is popular for its small size and low cost. Figure 2.11 shows the USB-equipped Digispark ATtiny85 microcontroller used in this thesis (left) and its length and width (right) measured using the iPhone's Measure app. As shown in Figure 2.11, the length of the

Table 2.3: systemd Unit Types [1]

Unit Type	File Extension	Description
Service Unit	.service	A system service, defines how to manage service daemons controlled by systemd.
Socket Unit	.socket	An interprocess communication socket, contains information about local interprocess communication or network sockets in Linux systems.
Target Unit	.target	A group of systemd units, used to provide synchronization between unit files during startup, and can also be used as a medium for extending the scope by specifying targets as other targets.
Device Unit	.device	A device file recognized by the kernel.
Mount Unit	.mount	A mount point for attaching filesystems controlled by systemd.
Automount Unit	.automount	A automount point for attaching filesystems controlled by systemd.
Timer Unit	.timer	A systemd timer, schedules the activation of other units.
Swap Unit	.swap	Describes a memory swap partition, can be a swap device or a swap file.
Path Unit	.path	A file or directory in a file system, defines paths that activate other services when files in the path are modified.
Slice Unit	.slice	A group of hierarchical organizational units for managing system processes, used in conjunction with cgroups to group processes, daemons, services into a hierarchical tree to manage resources.
Scope Unit	.scope	An externally created process.
Snapshot Unit	.snapshot	A saved state of the systemd manager.

ATtiny85 microcontroller is approximately 2cm, and the width is slightly less than 2cm.

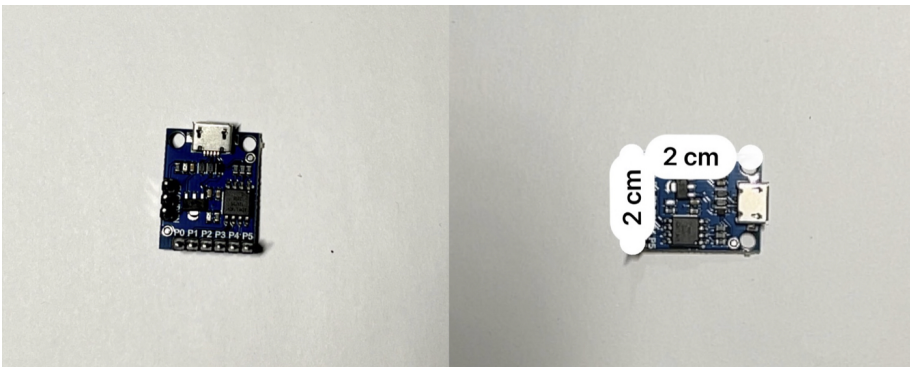

Figure 2.11: ATtiny85 (left) and Its Dimensions (right)

The ATtiny85 is the smallest microcontroller in the ATMEL AVR family, with a lower pin count for connecting different devices or sensors [67]. It has eight pins, and six are used for programmable input/output (I/O) [66]. Although it only has eight pins, it can perform almost everything a simple microcontroller can. For example, in this thesis, we use its security features to make it act as a watchdog timer. Moreover, it has utility in automation and other embedded systems, making it flexible and reliable.

2.4 Snort

Snort is a powerful open-source Intrusion Detection System (IDS) and Intrusion Prevention System (IPS) that provides real-time network traffic analysis and packet logging [2]. Snort enables network administrators to detect a variety of attacks and probes, such as Denial of Service (DoS) and Distributed DoS (DDoS) attacks, Common Gateway Interface (CGI) attacks, buffer overflows, stealth port scanning, etc. Snort creates a series of rules to define malicious network activity, identify malicious packets and send alerts to users. Furthermore, Snort can be used as a packet sniffer like tcpdump, a packet logger, a network file logging device, or a robust network intrusion prevention system [2].

As shown in Table 2.4, Snort has three different modes, i.e., packet sniffer, packet logger, and Network Intrusion and Prevention Detection System (NIPDS).

Table 2.4: Snort Modes [2]

Snort Mode	Description
Packet Sniffer	Read IP packets and display them to the user on its console.
Packet Logger	Log all IP packets accessing the network, so that network administrators can check who has accessed their network, and the operating system and protocol they are using.
NIPDS (Network Intrusion and Prevention Detection System)	Packets that are considered malicious are logged according to the preset characteristics of malicious packets defined in Snort rules, and actions are taken according to the rules set by the network administrator.

2.4.1 Snort Rules

Snort uses a rule-based language to perform protocol analysis and content search/matching. The rule syntax includes the rule headers and rule options. In this thesis, we run Snort in inline mode and use the Snort rules built by ourselves to detect and prevent DoS attacks at the network and transport layers.

Snort rule headers include rule actions, protocols, IP addresses, port numbers and direction operators [2, 3].

- The rule action tells Snort what to do when it finds a packet that matches the rule conditions. Table 2.5 summarizes the Snort actions.
- Snort currently analyzes four protocols for suspicious behaviour, i.e. TCP, UDP, ICMP, and IP. There may be more in the future, such as ARP, OSPF, RIP, etc.
- The next part of the rule header deals with the IP address for a given rule. Snort does not provide a mechanism for hostname lookup for IP address fields in the configuration file. An address consists of a straight numeric IP address and a CIDR block.
- The next part is port information for a given rule. Port numbers can be specified in several ways, including ports, static port definitions, ranges, and negation.
- Direction operator -> Indicates the direction of the traffic to which the rule applies. The IP address and port number on the left side of the direction

Table 2.5: Snort Rule Actions [2, 3]

Rule Action	Description
alert (default action)	Generate alerts using the selected alerting method, and log packets.
log (default action)	Log the packet.
pass (default action)	Ignore the packet.
drop (with inline mode)	Block and log the packet.
reject (with inline mode)	Block and log the packet. If the protocol is TCP, send a TCP reset message; if the protocol is UDP, send an ICMP port unreachable message.
sdrop (with inline mode)	Block the packet but do not log it.
Note: we can also define our own rule types and associate one or more output plugins with them.	

operator are considered to be the traffic from the source host, while the address and port number on the right side of the direction operator is the destination host.

Rule options form the core of Snort's intrusion detection engine, which is easy to use, powerful, and flexible. The semicolon separates the Snort rule options (;) and rule option keywords are separated from their arguments by a colon (:). There are four Snort rule options categories: general, payload, non-payload and post-detection. In this thesis, we focus on the general and post-detection rule options. The main rule options we use in this thesis are described below [3].

- **msg:** msg belongs to the general category, the message sent to the sysadmin if the rule is triggered.

- **sid:** sid belongs to the general category, used to identify Snort rules uniquely. This information enables the output plugin to identify the rules easily. In addition, this option should be used with the rev keyword. The SIDs less than 100 are reserved for future use; between 100 and 999999 are rules included in the Snort distribution; SIDs greater than 999999 are used for local rules.

- **rev:** rev belongs to the general category. The rev keyword is used to identify revisions of Snort rules uniquely. Revisions, along with Snort rule ID, allow signatures and descriptions to be refined and replaced with updated information. This option should be used with the sid.

- **classtype:** classtype belongs to general category. The classtype is used to classify the rule as detecting attacks belonging to a more general type of attack class. Snort provides a default set of attack classes to be used by the default set of rules. The default attack classifications are listed in Table 2.6 and Table 2.7.

- **detection_filter:** detection_filter belongs to the post-detection category. Defines a rate that a source or destination host must exceed before a rule can generate an event.

  - **Track <by_src / by_dst>:** rate is tracked either by source IP address or destination IP address, which means count is maintained for each source's unique IP address or each destination IP address. If we have multiple source hosts like DDoS, we need to track by destination.
  - **count <c>:** The maximum number of rule matches in s seconds allowed before the detection filter limit to exceeded. C must be nonzero.
  - **seconds <s>:** Time over which count is accrued. The value must be nonzero.

2.4.2 Snort inline mode

The Snort 4.0 package offers a new mode of operation called Inline IPS Mode, which we use in our thesis. Snort inline IPS Mode allows drop rules to trigger. In inline mode, Snort builds a bridge between two network segments and is responsible for passing traffic between the network segments. It can inspect the traffic it passes through and drops suspicious traffic.

Table 2.6: Snort Attack Classifications [4]

Classtype	Description	Priority
attempted-admin	Attempted administrator privilege gain	High
attempted-user	Attempted user privilege gain	High
inappropriate-content	Inappropriate content was detected	High
policy-violation	Potential corporate privacy violation	High
shellcode-detect	Executable code was detected	High
successful-admin	Successful administrator privilege gain	High
successful-user	Successful user privilege gain	High
trojan-activity	A network Trojan was detected	High
unsuccessful-user	Unsuccessful user privilege gain	High
web-application-attack	Web application attack	High
attempted-dos	Attempted DoS	Medium
attempted-recon	Attempted information leak	Medium
bad-unknown	Potentially bad traffic	Medium
default-login-attempt	Attempt to login by a default username and password	Medium
denial-of-service	Detection of a DoS Attack	Medium
misc-attack	Misc attack	Medium
non-standard-protocol	Detection of a non-standard protocol or event	Medium
rpc-portmap-decode	Decode of an RPC Query	Medium
successful-dos	DoS attack	Medium
successful-recon-largescale	Large scale information leak	Medium
successful-recon-limited	Information leak	Medium

Table 2.7: Snort Attack Classifications (cont'd.)

Classtype	Description	Priority
suspicious-filename-detect	A suspicious filename was detected	Medium
suspicious-login	An attempted login using a suspicious username was detected	Medium
system-call-detect	A system call was detected	Medium
unusual-client-port-connection	A client was using an unusual port	Medium
web-application-activity	Access to a potentially vulnerable web application	Medium
icmp-event	Generic ICMP event	Low
misc-activity	Misc activity	Low
network-scan	Detection of a network scan	Low
not-suspicious	Not suspicious traffic	Low
protocol-command-decode	Generic protocol command decode	Low
string-detect	A suspicious string was detected	Low
unknown	Unknown traffic	Low
tcp-connection	A TCP connection was detected	Very low

Chapter 3

Literature Review

This chapter discusses the related research on IoT security approaches against DoS attacks. Following a comprehensive evaluation of recent publications in this area, we have highlighted the main contributions of each related paper and compared them with our thesis. Table 3.1 and Table 3.2 show the contribution of this paper compared with other research papers in this area. In addition, we also present other possible research on IoT security solutions related to our thesis.

3.1 Comparative Study of Current Security Schemes for DoS Attacks

Since the increasing number of IoT devices and the latency-related issues in accessing, processing and storing data using cloud computing in IoT systems, Ullah et al. [68] propose a method for detecting and preventing internal and external DoS attacks in the fog layer of IoT systems, i.e., alight method. The proposed method is designed to protect fog-based alarm systems using fewer resources. This method is able to identify SYN flood attacks in the fog computing layer in real-time and avoid the attack by adding the intruder's device information to the blocking table (i.e. blacklist). However, the proposed method only detects and protects against one kind of DoS attack.

Simadiputra and Surantha [69] propose a secure and efficient Raspberry-Pi-based gateway for smart home IoT architecture, namely Rasefiberry. The proposed gateway has a security application and an intelligent encryption algorithm for files. In addition, the researchers use Snort as the intrusion detection system (IDS) of the proposed gateway and openHab as its application. The researchers evaluate Snort's ability to detect cyber DoS attacks and compare the efficiency of the well-known encryption algorithms. The results show that the Rasefiberry architecture can successfully handle the lightweight security program and the Openhab smart home gateway program. The Snort IDS proposed in this paper can only detect DoS attacks against the IoT

Table 3.1: Comparison of Raspberry House with Other Related Research

Research Work		Raspberry House	Ullah et al. [68]	Simadiputra & Surantha [69]	Aung & Thant [30]
IDS in IoT	Real Time Detection	C	C	C	C
	IDS Against System Level DoS Attack	C	×	×	×
	IDS Against Data Link Layer DoS Attack	C	×	×	C
	IDS Against Network Layer DoS Attack	C	×	C	×
	IDS Against Transport Layer DoS Attack	C	C	C	×
IPS in IoT	Real Time Prevention	C	C	×	×
	IPS Against System Level DoS Attack	C	×	×	×
	IPS Against Data Link Layer DoS Attack	C	×	×	×
	IPS Against Network Layer DoS Attack	C	×	×	×
IPS in IoT	IPS Against Transport Layer DoS Attack	C	C	×	×
	IPS Against Transport Layer DoS Attack	C	C	×	×
IDS/IPS Design for Internal DoS		C	C	C	C
Alert User Approach		C	×	×	×
IDS/IPS Built for Resources Constraint IoT Devices		C	×	C	×
×: Not Covered			C: Covered		

Table 3.2: Comparison of Raspberry House with Other Related Research (cont'd.)

Research Work		**Zhang & Sampalli [53]**	**Nguyen et al. [70]**	**Java-nmardi et al. [71]**	**Binu et al. [72]**	**Mariam & Negash [73]**
IDS in IoT	**Real Time Detection**	×	C	C	C	×
	IDS Against System Level DoS Attack	×	×	×	×	×
	IDS Against Data Link Layer DoS Attack	×	×	×	C	×
	IDS Against Network Layer DoS Attack	×	×	×	C	×
	IDS Against Transport Layer DoS Attack	×	×	C	C	C
IPS in IoT	**Real Time Prevention**	C	×	C	C	×
	IPS Against System Level DoS Attack	×	×	×	×	×
	IPS Against Data Link Layer DoS Attack	C	×	×	C	×
	IPS Against Network Layer DoS Attack	×	×	×	C	×
	IPS Against Transport Layer DoS Attack	×	×	C	C	×
IDS/IPS Design for Internal DoS		C	C	×	×	×
Alert User Approach		C	×	×	×	×
IDS/IPS Built for Small IoT Devices		C	×	×	C	×
×: Not Covered			C: Covered			

network layer and transport layer protocols, and the authors do not provide IPS in their proposed IoT gateway to prevent DoS attacks.

Kristiyanto et al. [52] use external penetration testing to analyze deauthentication attacks on IoT-based IEEE802.11 connections, aiming to analyze the security level of IEEE802.11 (i.e. WiFi) connections against IoT device-based deauthentication attacks. The results show that the deauthentication attacks result in a loss of communication between the test target and the gateway, but the test target's media access control (MAC) is still registered on the gateway. In this paper, the authors list some mitigation strategies, but they do not apply them in this paper.

The link layer in wireless networks is vulnerable to DoS attacks due to unprotected management frames. Therefore, Aung and Thant [30] propose an approach to detect and mitigate the wireless link layer attack (i.e., the WLLA algorithm), aiming to detect masqueraded DoS attacks using active and passive fingerprinting methods. The proposed algorithm is implemented in a real-time setup using Python network programming under the Kali Linux environment. The results show that the proposed algorithm can detect wireless link layer DoS attacks. Furthermore, the authors indicate that they would consider using the prevention and auditing modules for management frames in their future research.

Zhang and Sampalli [53] propose a client-based IPS for IEEE802.11 wireless LAN to prevent Mac layer DoS attacks. The proposed system helps clients distinguish between legitimate and forged management frames using a MAC filtering mechanism. Moreover, the proposed solution is a lightweight solution that does not require significant changes to existing standards or the use of cryptography. Although the proposed scheme can protect wireless clients from DoS attacks of management frames initiated at the MAC layer, the authors did not consider the IDS and IPS approach for other DoS attacks.

Nguyen et al. [70] propose a lightweight DNN-based Network Intrusion Detection System (NIDS) (i.e., Realguard) for IoT gateways, aiming at running directly on local gateways and protecting internal IoT devices. The proposed system can accurately detect multiple network attacks in real-time with a small amount of computation. However, the proposed attack detection model must be frequently retrained to maintain high accuracy, which will consume a lot of computational and network resources

to deploy and update the model.

Javanmardi et al. [71] propose a security-driven task scheduling method (i.e., FUPE) for SDN-based IoT-Fog networks. FUPE is a security-aware task scheduler in IoT fog networks, which aims to efficiently assign IoT end-user tasks to fog devices in SDN architecture. The results show that FUPE can optimize network and computer resource utilization while blocking TCP SYN flood attacks to prevent resource exhaustion. Although the authors only consider TCP SYN flood attacks in this paper, in future work, the authors show that they will use FUPE to prevent other DoS attacks such as ICMP flood attacks.

Binu et al. [72] propose an SDN-based IoT DoS attack dynamic detection and mitigation scheme to detect three common DoS attacks: deauthentication, TCP SYN flood, and Ping of Death. The authors use Mininet to simulate an entire system capable of detecting highly accurate attacks using the SVM algorithm and mitigating malicious traffic within seconds by implementing new rules in the SDN controller. The proposed system effectively mitigates DoS attacks in the TCP/IP model. However, DoS attacks that occur at the system level are not considered.

Mariam and Negash [73] evaluate the performance of machine learning algorithms to detect SYN flood attacks. The classification model was trained and tested using a packet capture dataset collected from the Ethiopian Telecom network by generating and capturing packets using the Hping3 and Wireshark. The authors use four classification machine learning algorithms to test: Naive Bayes, Adaptive Booster (AdaBoost), J48, and Artificial Neural Network (ANN). The results show that the J48 model can detect SYN flood attacks more effectively than the other three models. The authors conclude that they will study IDS and IPS methods for DoS attacks against other protocols in future work.

3.2 IoT Gateway Design

Shang et al. [74] propose a novel configurable smart IoT gateway. The proposed smart IoT gateway can support Ethernet, 3G and RS485 bus for data communication with the common network. Users can choose different data communication interfaces according to their specific needs. In addition, the gateway has unified external interfaces, which are suitable for flexible software development. It has flexible

protocols to convert different sensor data into a uniform format. The results show that the proposed gateway has better scalability and flexibility, and it is a low-cost gateway. However, the proposed gateway does not support WiFi communication, and it does not include any security policy.

Glória et al. [34] propose an IoT gateway for creating a smart swimming pool dedicated to real-time monitoring and remote control of a swimming pool. The proposed gateway is designed using Raspberry Pi. The gateway allows bidirectional communication and data exchange between the user and the sensor network implemented in the environment using Arduino. However, the proposed gateway only serves a specific application and does not provide security functions.

Zachariah et al. [75] propose an IoT gateway architecture that uses a generic IoT gateway as a software service on modern smartphones to provide generic and ubiquitous Internet access to Bluetooth Low Energy (BLE)-connected IoT devices. The proposed IoT gateway uses wireless communication protocols for specific applications, which provides a scalable alternative to application-specific gateway structures. The proposed IoT gateway utilizes smartphones as IPv6 routers for less resource constrained endpoints and as a BLE proxy.

3.3 IDS for DoS/DDoS Attacks in IoT Security

Chen et al. [76] propose an ML-based IoT DDoS attack detection system to prevent DDoS attacks in IoT gateways. The proposed system is a multilayer DDoS detection system including IoT devices, IoT gateways, SDN switches and cloud servers. The authors extract the characteristics of four DDoS attacks and launch DDoS attacks from eight smart poles. The research results show that the proposed system can detect DDoS attacks and block malicious devices.

Gupta et al. [77] propose a machine learning-based attack detection method to identify attack traffic in the Consumer Internet of Things (CIoT). The proposed method uses a low-cost machine learning classifier to detect attacks on the local router. The authors record normal and attack traffic patterns by simulating IoT networks and use six machine learning classifiers to detect them based on the prepared dataset. The results show that the proposed method achieves an accuracy of 0.97–0.99, and it also acts as a mitigation strategy by filtering out attack traffic on the local router.

Ferrag et al. [78] propose a DDoS attack IDS based on three deep learning models, which are convolutional neural network (CNN), deep neural network (DNN) and recurrent neural network (RNN), aiming to provide performance evaluation and comparative analysis between machine learning and deep learning methods in Agriculture 4.0 network. The authors use the CIC-DDoS2019 dataset and the TON IoT dataset and study the performance of each model inside two classification types. The results show that deep learning techniques are more accurate than other machine learning strategies.

Nie et al. [79] propose a secure social IoT intrusion detection based on collaborative edge computing. The authors use a feature selection module to process collaborative edge network traffic and design an intrusion detection system for a single attack based on a generative adversarial network (GAN). It has been verified that the proposed method can effectively perform intrusion detection.

SaiSindhuTheja and Shyam [80] propose an efficient meta-heuristic OCSA algorithm based on feature selection and Recurrent Neural Network (RNN) for DoS attack detection in a cloud computing environment. The proposed algorithm combines opposition-based learning (OBL) and the crow search algorithm (CSA), selects the feature, and then uses RNN to classify the selected features. The authors use the KDD Cup 99 dataset to conduct experiments. The experimental results show that the proposed algorithm is better than existing algorithms in terms of Precision, Recall, F-Measure and Accuracy.

Since DDoS attacks are one of the major security threats to cloud computing, Velliangiri et al. [81] propose a DDoS attack detection scheme to detect intruder nodes in cloud environments. The proposed solution is based on a Taylor-Elephant Herd optimized Deep Belief Neural Network (TEHO-DBN) classifier and compared with Neural Network (NN), Ensemble, Support Vector Machine (SVM), Elephant Herd Optimisation (EHO) algorithms. The results show that the proposed TEHO-based Deep Belief Network (DBN) classifier improves the performance with an accuracy of 0.83.

Sousa et al. [82] propose IDS-IoT, an IDS for IoT DoS attacks, designed to detect specific DoS attacks, including SYN flood, land attack, ICMP flood, smurf attack, and UDP flood. The results show that the proposed IDS-IoT achieves good results

in detecting the types of attacks implemented, with no false-positive events observed during testing. The limitations of IDS-IoT are that it needs to capture a large number of packet characteristics for analysis, which leads to high operating costs and limited attack types detected.

3.4 IPS for DoS/DDoS Attacks in IoT Security

Mihoub et al. [83] propose an architecture for DoS/DDoS attack detection and mitigation in IoT using the looking-back-enabled machine learning technique to detect and mitigate DoS/DDoS attacks in IoT. The proposed architecture is able to detect and mitigate DoS/DDoS attacks against TCP, UDP and HTTP protocols and is evaluated on the Bot-IoT dataset. The evaluation results show that the random forest classifier with looking-back-enabled achieves 99.81% accuracy.

Saxena and Dey [84] propose a data packet tracing approach based on a third-party auditor (TPA) to prevent DDoS attacks using a collaborative cloud computing approach. The authors use the Weibull distribution to analyze the source of DDoS attacks and obtain the availability, reliability, and median lifetime of DDoS defence in a cloud environment. Moreover, the proposed approach also solves the problem of IP spoofing. The authors use an application based on the Hadoop and MapReduce framework to test this approach. The proposed approach can effectively mitigate and prevent DDoS attacks compared with other existing approaches. In addition, the proposed approach reduces the overhead of cloud users.

Kumar and Amin [85] propose a blockchain and software-defined networking (SDN) approach to mitigate DDoS attacks. SDN's ability to fully authenticate and filter traffic provides an appropriate mechanism for authenticating legitimate users. In addition, SDN also has the ability to enforce rules on the SDN controller. For example, if the IP address is not authenticated, the SDN drops all requests from that IP address, which is very helpful in mitigating attacks. The results show that the proposed solution can effectively mitigate DDoS attacks.

3.5 Research Gaps

In this related work section, we found many IoT gateways that lack security functions, making them vulnerable to attack. Many techniques exist to identify IoT security vulnerabilities, learn attacker behaviour, and continuously monitor IoT devices to detect and prevent DoS attacks against IoT devices. However, some approaches do not seem to be generic enough to address the heterogeneity of the IoT paradigm. Furthermore, although IDS techniques in the IoT field demonstrate advanced progress, some of these methodologies leave room for further research. Indeed, relying only on IDS mechanisms to monitor DoS attacks is not very effective, as they can only detect limited attacks. Therefore, the research community has received attention from the study of how to provide effective IPS technologies. However, the existing research on IPS technologies for DoS attacks is far less than the research on IDS technologies because the heterogeneity of the IoT paradigm makes it impossible to implement the complex IPS technology on resource-constrained IoT devices.

In addition, we found that most of the IDSs and IPSs for IoT DoS attacks are based on machine learning approaches. The proposed machine learning approaches first need to capture normal and malicious packets and construct a dataset for training purposes. They will then train the dataset using various machine learning algorithms to improve the accuracy of capturing malicious packets (i.e., improving the IDS accuracy). Therefore, researchers spend a lot of time upfront to generate the dataset and select parameters to optimize algorithms, which will result in high operating costs, time costs and resource usage. Furthermore, many IDS and IPS approaches can only be run once, which is not perfect for IoT gateways: we need the proposed IDS and IPS to run automatically when the IoT gateway is powered and easy to operate.

3.6 Contributions

The main contribution of this thesis is to provide researchers with a novel approach for intrusion detection and prevention of DoS attacks on IoT devices. We added security functions to the proposed IoT gateway (i.e. Raspberry House) to improve its security. In addition, the proposed intrusion detection and prevention approaches are compatible with resource-constrained IoT devices. The proposed IDS and IPS

methods are mainly based on shell scripts and systemd services, and take into account DoS attacks in various scenarios, including but not limited to SYN flood attacks, ICMP flood attacks, Deauthentication attacks, and bash fork bomb. Since no training on a dataset is required, our new method requires less time cost than the machine learning approach. Moreover, since we use systemd services, the proposed IDS and IPS methods can run automatically after powering up the Raspberry House. We tested the average delay of the proposed IDS and IPS methods at a given time for reference by other researchers. Finally, we provide researchers with suggested optimal IPS solutions for different DoS attacks for their further study.

Chapter 4

Proposed Raspberry House Design

This chapter proposes a general gateway framework for IoT systems that can provide a secure communication environment for small IoT devices connected to it. In addition, the proposed gateway is able to detect and prevent different kinds of DoS attacks and alert the administrator in real-time. The Raspberry House gateway design in this thesis consists of three parts. The first part is the architecture design of Raspberry House; the second part is the design of IDS and IPS in Raspberry House; the third part is the testing of Raspberry House IDS and IPS security based on Kali Linux tools; the last part summarizes the advantages of the designed architecture.

4.1 Raspberry House Architectural Design

Our design goal is to design an IoT gateway with accessibility and low cost. The proposed Raspberry House is designed for small IoT devices to provide a secure network environment for the devices connected to it and to be able to detect and prevent internal DoS attacks. In addition, in order to improve its utilization, Raspberry House has the characteristics of small size and easy portability.

Figure 4.1 shows the architectural design of the Raspberry House. We have configured the Raspberry House as an IoT gateway in our previous research [63], which can separate the internal network from the external network, i.e. generate private network IP addresses and assign them to small IoT devices connected to it. Therefore, all the small IoT devices connected to it are in a private network, and there is no way for external networks to bypass the Raspberry House and connect to the small IoT devices.

The Raspberry House authentication feature integrates IoT gateway authentication and logging into the Raspberry House using SSH. Users from the Internet or small IoT devices must enter their user credentials, i.e., user ID and password, to access and connect to the gateway. If user credential verification fails, the user must

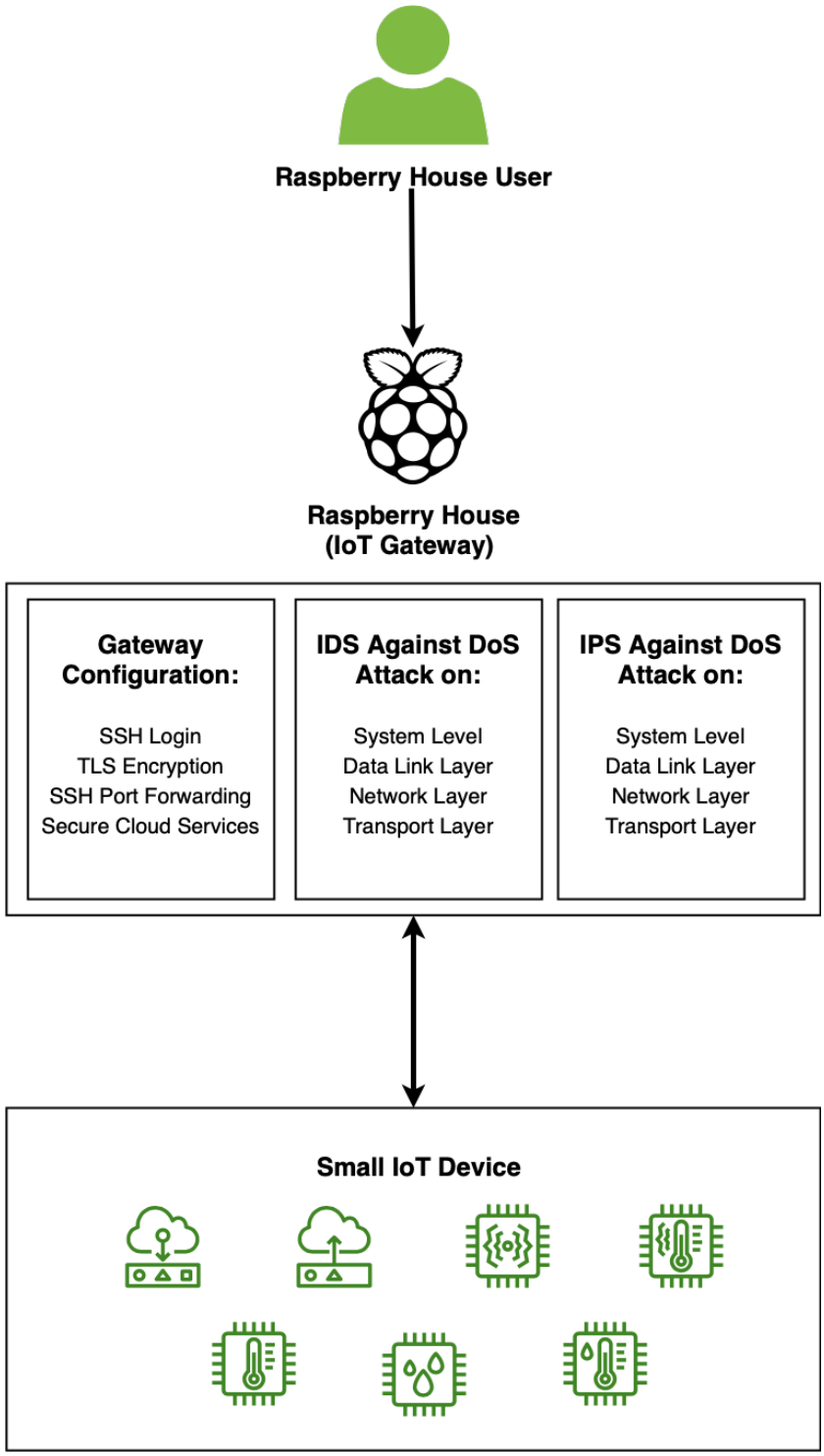

Figure 4.1: Raspberry House Architecture Design

enter the password again until they enter the correct password and user ID. Furthermore, the administrator can log into the Raspberry House system using SSH with their hostname and password, therefore, the administrator does not need to login to Raspberry House using an external monitor, keyboard and mouse.

The Raspberry House uses the TLS protocol so that every message sent will pass a message authentication code (MAC) to check the integrity of the message. Moreover, the protocol can use the public key to verify the identity of the communicating party and, if necessary, the identities of both parties. In addition, considering that Raspberry House may manage a large number of IoT devices in the future, Raspberry House also provides a secure cloud service.

This research mainly focuses on the design of Raspberry House's intrusion detection and prevention system, which aims to protect IoT devices from DoS attacks. We introduce the design of IDS and IPS in the next section.

4.2 Raspberry House IDS and IPS Design

The intrusion detection and prevention system we designed can operate autonomously and generate real-time feedback when the Raspberry House is powered, which means that even if the administrator does not log into the Raspberry House, the IDS and IPS can still operate. Since all the IDS and IPS we designed are based on shell scripts, in order to be able to make them run autonomously when the Raspberry House is powered on, we need to design the systemd services to start the IDS and IPS services in parallel. We divide the IDS design and IPS design into two parts for a detailed description.

4.2.1 Raspberry House IDS Design

For the IDS at the data link layer, we wrote a Python script in a Linux environment designed to monitor all wireless traffic to detect deauthentication attacks at the data link layer. Before running this Python script, we need a shell script to start the monitor mode of the Raspberry House and read all network traffic of the wireless devices connected to it and all interfaces. Algorithm 1 shows the algorithm of the proposed Python script to detect deauthentication packets. The Python script first continuously captures real-time packets to detect any data transfer or network activity. The

captured traffic is then analyzed. If a WiFi (IEEE802.11) packet in the data link layer is captured, the capture date-time is recorded and converted into a UNIX timestamp, packet type, and packet subtype. If the packet type is a management frame and the packet subtype is a deauthentication frame, then the script detects a deauthentication packet. Furthermore, we can detect other DoS attacks at the data link layer, such as disassociation attacks by changing the subtype of the detect packets in the Python script. Finally, the script prints out the captured packet information for subsequent use.

Algorithm 1 Detect Deauthentication Packets

Input: Run the Python script.

Output: the date time, UNIX timestamp and attacked MAC address if deauthentication attack detected.

1: Configure monitor mode
2: Detect data link layer attack()
3: Monitor Packets (p)
4: **if** Monitored packets is in Layer of Dot11 **then**
5: Formate datetime: $formattime$
6: Convert datetime to UNIX timestamp: $unixtime$
7: type = p.getlayer(Dot11).type
8: subtype = p.getlayer(Dot11).subtype
9: **end if**
10: **if** type==0 and subtype==12 **then**
11: Set $macaddr$ equal to the attacked MAC address with all capital letter
12: Print $formattime$ + $unixtime$ + "Deauth Detect Against Mac Addr" + $macaddr$
13: **end if**

After a deauthentication packet is detected, we will publish the packet information on a topic using the MQTT broker in one shell script and then use the pub/sub mode to subscribe to the same topic and get the detected packet information in another shell script. Moreover, for security, we use the local MQTT broker to publish and subscribe messages to reduce the risk of eavesdropping. If a message is received from an MQTT publisher, then a malicious packet has been detected, so our IDS

will generate an alert message and send the alert to the administrator via email or Twitter message. In addition, a LED connected to the Raspberry House will blink to alert the user that an intrusion has been detected.

For network and transport layers IDS, we use the Snort ruleset to detect the DoS attacks, which can reduce processing speed and RAM usage load. Snort rules are divided into three categories: community ruleset, subscriber ruleset, and custom ruleset (i.e., local rules). The community ruleset is a GPLv2 Talos certified ruleset, free and open to all users, without any Snort subscriber ruleset licensing restrictions. The community ruleset can detect most attacks on the network and transport layers, but it is relatively weak due to the simple parameter setting. Subscriber ruleset is developed, tested, and approved by Cisco Talos. Subscriber ruleset includes community ruleset and requires users to pay for use but will update rules without delay. Custom rules mean that users can define Snort rules for intrusion detection and prevention according to their own needs. The rules need to be defined by users without any cost. Since our research goal is to set up a secure and low-cost gateway, in this thesis, we use the free default Snort community rules and custom Snort rules to effectively detect DoS attacks against the network layer and transport layer.

Since Snort is able to detect suspicious behaviour for both network layer and transport layer protocols (i.e., IP, ICMP, TCP, and UDP), this thesis uses it as the IDS for the network and transport layers. We write a shell script to publish the Snort detection results to an MQTT topic through the local MQTT broker and create another shell script that acts as a subscriber to collect the information published on the topic. After detecting suspicious traffic, the publisher will publish it on the relevant topic, and after the subscriber receives the detection results, an alert will be generated and sent to the administrator through email and Twitter messages. Moreover, a LED connected to the Raspberry House will blink to alert the user that an intrusion has been detected.

We use the watchdog timer for system security level IDS to detect suspicious traffic. A Watchdog Timer (WDT) is used to monitor system programs to see if they are out of control or have stopped running [86, 87]. The Raspberry House communicates with the watchdog timer at a set interval to indicate that it is still working normally. If Raspberry House does not output a signal, outputs too many

signals, or outputs a different signal than the preset pattern, the watchdog timer will detect that the Raspberry House system is malfunctioning and send a reset signal to Raspberry House. The Raspberry Pi 3B+ offers a built-in watchdog timer service, but consider that there is no guarantee that a failing Raspberry House will be able to monitor itself and detect failures when it is only monitored by the built-in watchdog timer. Therefore, for the system level IDS design, we also propose the use of the external watchdog timer that runs independently, aiming to add an extra layer of security to the Raspberry House. This thesis uses the ATtiny85 board as the external watchdog timer. After a system anomaly is detected, administrators receive security alerts via email and Twitter messages for further action.

4.2.2 Raspberry House IPS Design

For the IPS design of the data link layer, for illustration purposes, we use the deauthentication attack as an example of a data link layer DoS attack. However, this IPS can be implemented on other DoS attacks against IEEE802.11 wireless networks, such as disassociation attacks. The deauthentication attack is a destructive technique against wireless connections that prevents IoT devices from connecting to the Raspberry House. Since the network does not validate incoming frames, if the attacker keeps sending deauthentication requests, the user cannot reconnect to Raspberry House for a long time.

A common way to prevent such an attack is to use Protected Management Frames (PMF, AKA 802.11w) [88], that is, add a hash and a signature to the management frame, but small IoT devices do not support 802.11w due to their limited resources. Therefore, we propose a shell script as IPS. When a deauthentication attack is detected, the IPS in the Raspberry House will be triggered if the deauthentication packets received by the Raspberry House exceed a defined threshold. The proposed IPS will block the WiFi interface and then re-enable the WiFi interface after a predefined interval. If the deauthentication packets are still received after unblocking the WiFi interface, then the IPS will repeat the above operation until the Raspberry House no longer receives any deauthentication packets. After our IPS blocks the Raspberry House's WiFi interface, the attacker cannot continue to send deauthentication packets to the target network because the attacker cannot find the target.

For the network layer and transport layer IPS, we use Snort's inline mode and custom ruleset for implementation. When Snort is in inline mode, it acts as an IPS, allowing drop rules to be triggered. In inline mode, Snort builds a bridge between two network segments and is responsible for passing traffic between the network segments. It can detect passing traffic, as well as suspicious drop traffic. In order to create a bridge between two network segments, Snort needs to have two network interfaces, each on different network segments. We will configure these interfaces in the promiscuous mode without IP addresses. When Snort is running, it will check the traffic on each interface, detect the packets against the rules we defined, drop malicious packets as defined in the rules, or send them to another interface without making any modifications. Therefore, Snort network segments must belong to the same logical subnet. Since the Snort application is responsible for bridging (i.e., passing traffic between the two network segments), if Snort is not running, the two network segments will not be able to communicate through the Snort system. The IPS we designed uses Snort inline mode and custom Snort rules to detect and drop malicious packets defined in the rules to protect small IoT devices connected to the Raspberry House from the DoS attack.

We use watchdog timers as the system security level IPS. After detecting an anomaly, the watchdog timer sends a reset signal to the Raspberry House to reboot it. For system security level DoS attacks, we have designed three watchdog timers, namely built-in watchdog timer (hardware watchdog) and external watchdog timer (gentle watchdog timer and delayed watchdog timer). Table 4.1 explains the proposed watchdog timer schemes. Figure 4.2 shows the workflow of the external watchdog timer (WDT). The built-in watchdog timer is achieved using the watchdog timer functionality provided in the Raspberry Pi SoC (system on a chip), and we will detail how we download and configure it to the Raspberry House in the next chapter. The built-in watchdog timer prevents attacks such as the bash fork bomb attack that are easily detected by the system and cause the system to slow down or crash. For the gentle watchdog timer and the delayed watchdog timer, we designed a shell script that will check the specified GPIO pins on the Raspberry House and run different watchdog timers depending on the state of the pins. The gentle watchdog timer can be used for DoS attacks that are not easily detected, such as attacks against inodes.

Table 4.1: Description of the Proposed Watchdog Timer (WDT) Schemes

WDT Type	Description
Built in WDT	When the DoS attacks make Raspberry House freeze and are easily detected, the hardware watchdog will restart the Raspberry House automatically after a predefined interval.
Gentle WDT	When DoS attacks which make Raspberry House freezes, but difficult to be detected immediately, the gentle WDT will restart Raspberry House automatically after a predefined interval.
Delayed WDT	The delayed WDT will run when the DoS attack is massive and sustained. It will restart the Raspberry House automatically after a predefined interval (the time interval will be relatively long compare to the other two WDT since it will restart the Raspberry House after the large amount DoS attacks).

The delayed watchdog timer can be used to detect massive and sustained DoS attacks. Table 4.2 summarizes the IDS and IPS designs proposed in this thesis.

4.3 Raspberry House IDS and IPS Evaluation Design

This thesis uses Kali Linux based tools to test the security of Raspberry House IDS and IPS. Figure 4.3 shows the evaluation design scheme for our research. We use Kali Linux based PC as the attacker for DoS attacks. Kali Linux comes pre-installed with most of the security applications for penetration testing. The applications used in this thesis are the aircrack-ng suite and hping3 command to run flood-type DoS attacks to test the IDS and IPS performance at the data link layer, network layer, and transport layer. For system security level IDS and IPS, we use the system level penetration test commands on the Raspberry House to test.

4.4 Advantages of The Designed Architecture

The following are the advantages of the designed architecture proposed in this thesis.

- The proposed Raspberry House can provide a secure environment for IoT researchers/engineers.

Table 4.2: Proposed Raspberry House's IDS and IPS Design

DoS Attacks on...	IDS Design	IPS Design
Data Link Layer	- Develop a Python script to detect the DoS attacks against IEEE802.11 network. The malicious packets can be detected based on the type and subtype of their frames. - Blink LED to notify the user. - Send alert through email and twitter to the user.	Block the WiFi interface for a predefined interval and then unblock the WiFi interface. Repeat the above operation until the Raspberry House no longer receives any malicious packets.
Network Layer & Transport Layer	- Use community ruleset and custom rules to detect the attacks. - Blink LED to notify the user. - Send alert through email and twitter to user.	Use inline mode and custom rules to drop the malicious packets.
System Security Level	- Design different kinds of WDT (built-in WDT, gentle WDT, and delayed WDT) to detect the DoS attacks based on the serious of the attacks. - Send alert through email and twitter to the user.	The WDTs will reboot Raspberry House with different predefined intervals to prevent DoS attacks.

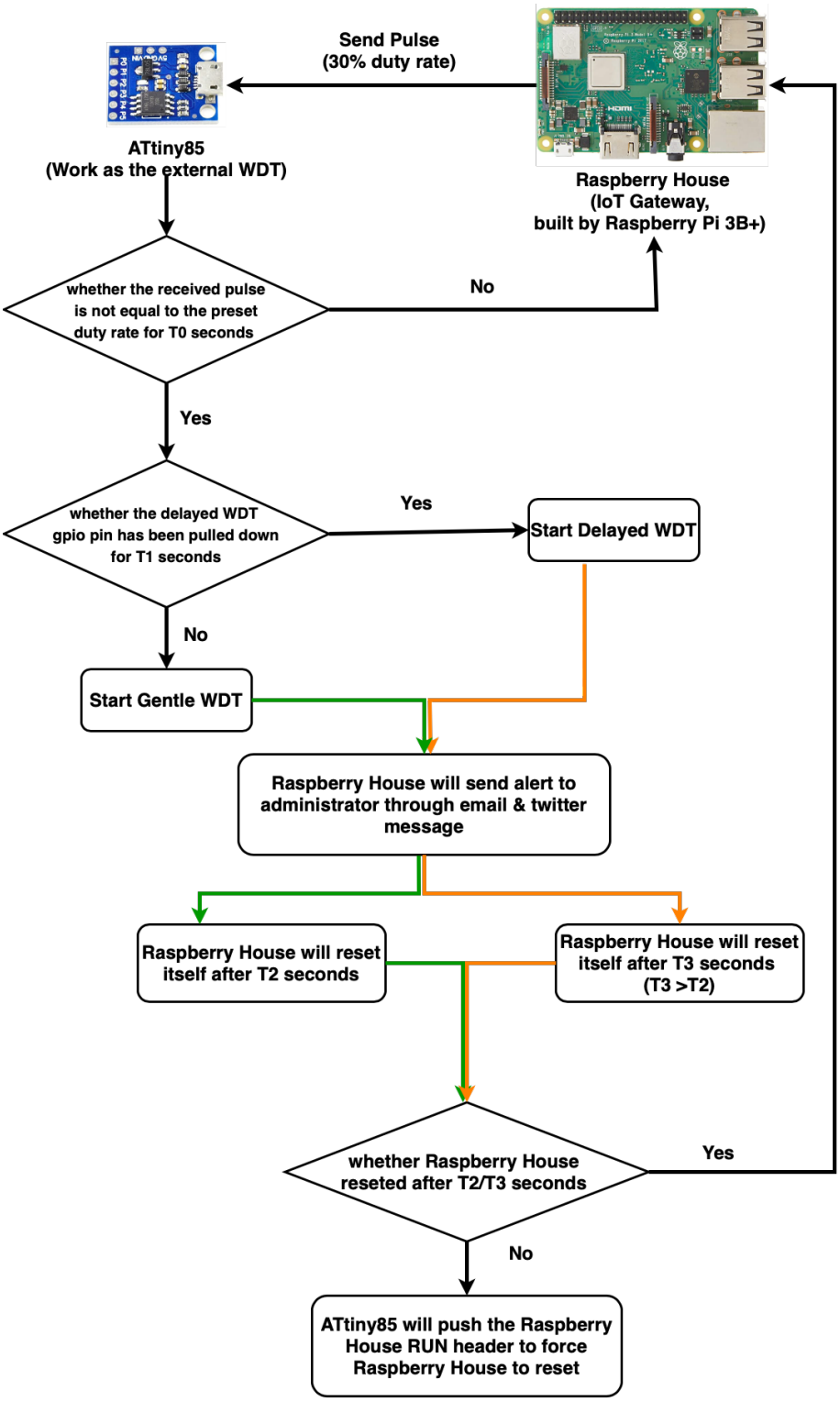

Figure 4.2: External WDT Workflow

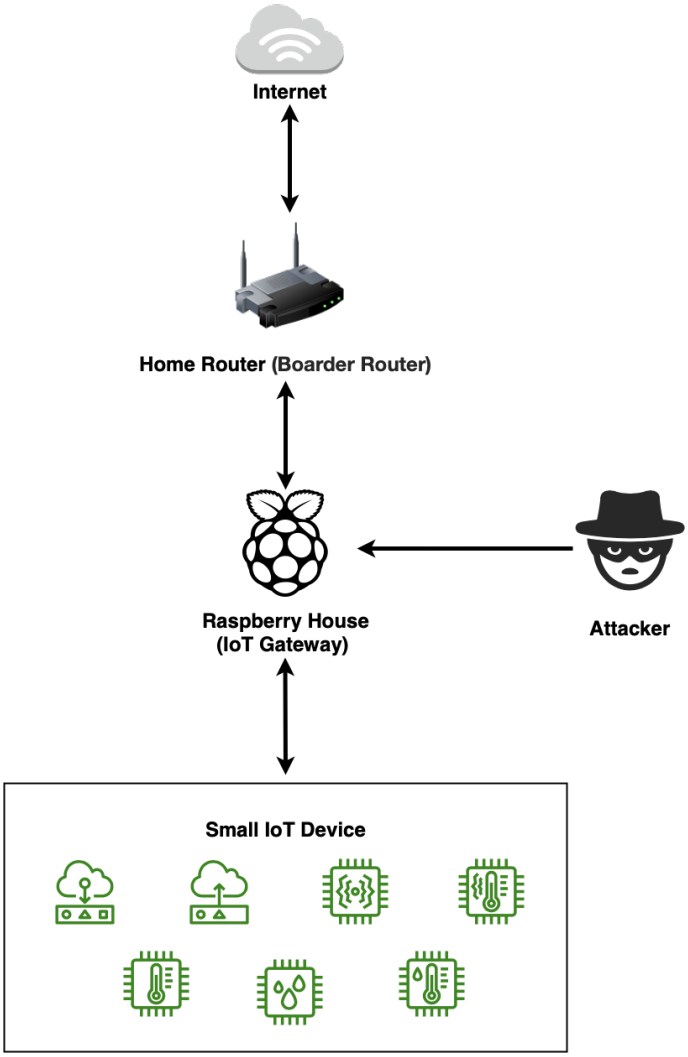

Figure 4.3: Raspberry House IDS and IPS Evaluation Design

- The proposed Raspberry House can detect and prevent DoS attacks in real-time to enhance the security of small IoT devices which connect to it.

- The proposed Raspberry House considers IDS and IPS against DoS attacks in the data link layer, network layer and transport layer of the TCP/IP model.

- Moreover, the proposed Raspberry House considers IDS and IPS against DoS attacks at the system security level.

- The proposed Raspberry House considers the severity of DoS attacks and decides on preventive measures based on severity, improving system efficiency.

- The proposed Raspberry House can send alerts to users through email and Twitter when a DoS attack is detected.
- The proposed Raspberry House is low cost and easy to carry to ensure that most IoT researchers can afford it and use it to work anywhere.

Chapter 5

Experimental Implementation

5.1 Raspberry House Architecture Implementation

Figure 5.1 shows the implementation of the proposed scheme. A Raspberry Pi 3B+, Ethernet cable, USB to Ethernet adapter, Micro SD card, SD card reader, universal PCB board, USB WiFi adapter and MacBook Pro (server) as the hardware to implement the Raspberry House; ATtiny85 board as the hardware implementing external watchdog timers (gentle WDT and delayed WDT); attacker using Raspberry Pi 4 with Kali Linux system as the malicious PC; NodeMCU board and Grove Beginner Kit for Arduino (GBKA) as our small IoT device. In addition, due to the resource constraints of the research, there is only one small IoT device, but Raspberry House can accept multiple Internet devices connecting to it.

As shown in Part A, we use MacBook Pro as a server and used the Ethernet interface of the Raspberry Pi to connect to the MacBook Pro with an Ethernet cable. The server uses a wireless connection to the home router in Part B. The home router is then connected to the Internet. In this way, we have made a gateway in the gateway, which enhances the security of small IoT devices connected to our gateway. In addition, as shown in Part C, we plugin the universal printed circuit board (PCB) onto the Raspberry House because we need to use the LEDs on it to indicate different DoS attacks. Universal PCB is an environmental sensor board module for Raspberry Pi (B+/2/3) that integrates Bosch's environmental sensor "BME280" and an illuminance sensor. There are also three push button switches, three LEDs, and an IR LED mounted on the board, which is handy when making a stand-alone device. Furthermore, it also has a terminal to connect to the "AQM1248A Small Graphic LCD Panel" as an option. As shown in Figure 5.2, we use the stacking headers to connect the universal PCB and Raspberry House. The upper blue colour board is the universal PCB, and the lower board is Raspberry House.

In Part D, we use the WiFi interface of the Raspberry House to connect small IoT

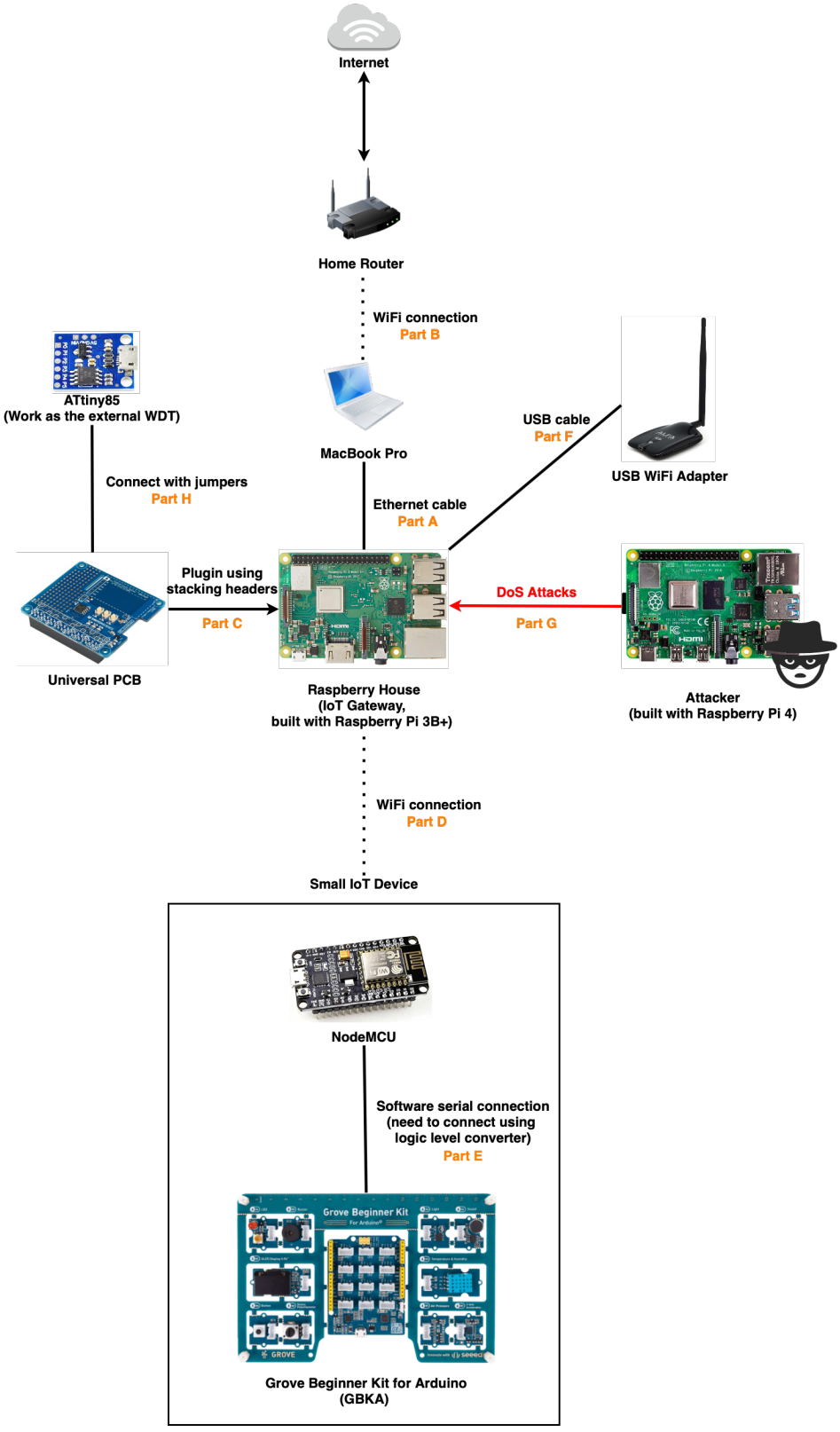

Figure 5.1: Connection of Raspberry House

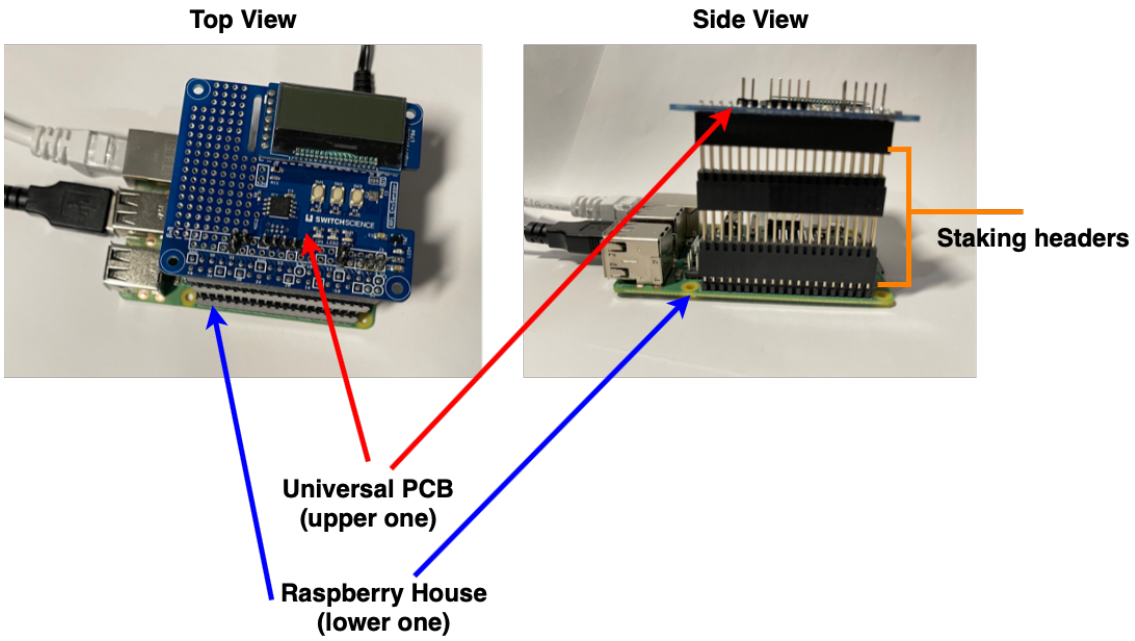

Figure 5.2: Connection Between Raspberry House and Universal PCB

devices. This thesis uses NodeMCU and Grove Beginner Kit (GBKA) as a small IoT device. GBKA is an all-in-one kit designed to provide users with an Arduino programming study. The kit is powered by an Arduino compatible board (Seeeduino Lotus) and ten additional Grove Arduino sensors, all integrated on one board, which fulfills our needs for different kinds of small IoT devices in this thesis. Figures 5.3 and 5.4 show the hardware overview of GBKA and its size in detail [9]. However, GBKA does not offer wireless connectivity. We need small IoT devices to be able to connect to the secure wireless network provided by the Raspberry House. Therefore, we use NodeMCU to achieve this purpose. NodeMCU board includes ESP8266 with a WiFi-enabled chip. ESP8266 is a low-cost WiFi chip developed using the Espressif system's TCP/IP protocol. Furthermore, both GBKA and NodeMUC can be programmed using the Arduino IDE. Part E shows that NodeMCU and GBKA are connected via software serial to transmit data. Since GBKA uses 5v and NodeMCU uses 3v3, we need a logic level converter for voltage conversion. The specific connections are shown in Table 5.1.

As shown in Part F, the USB WiFi adapter we use to monitor the network traffic is Alfa AWUS036NHA Wireless B/G/N USB Adaptor, and we connect it to Raspberry House using a USB cable. In Part G, we use Kali Linux based Raspberry Pi 4 as the

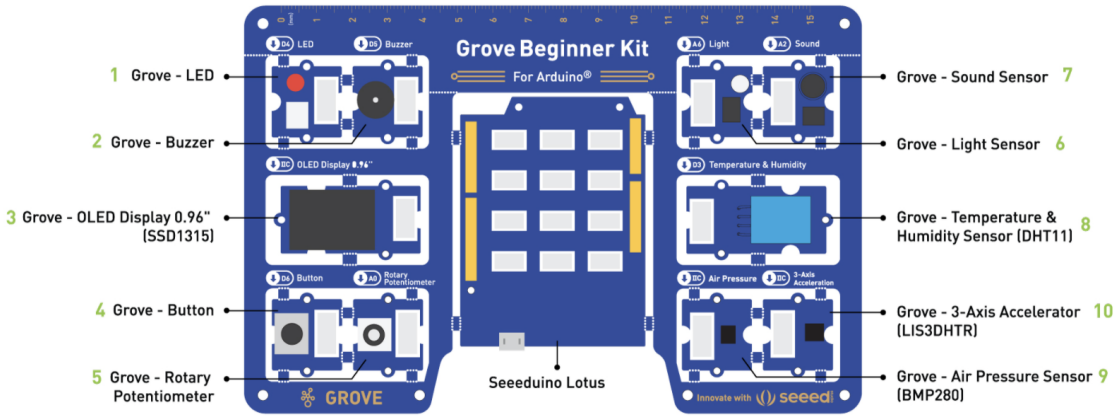

Figure 5.3: Hardware Overview of GBKA [9]

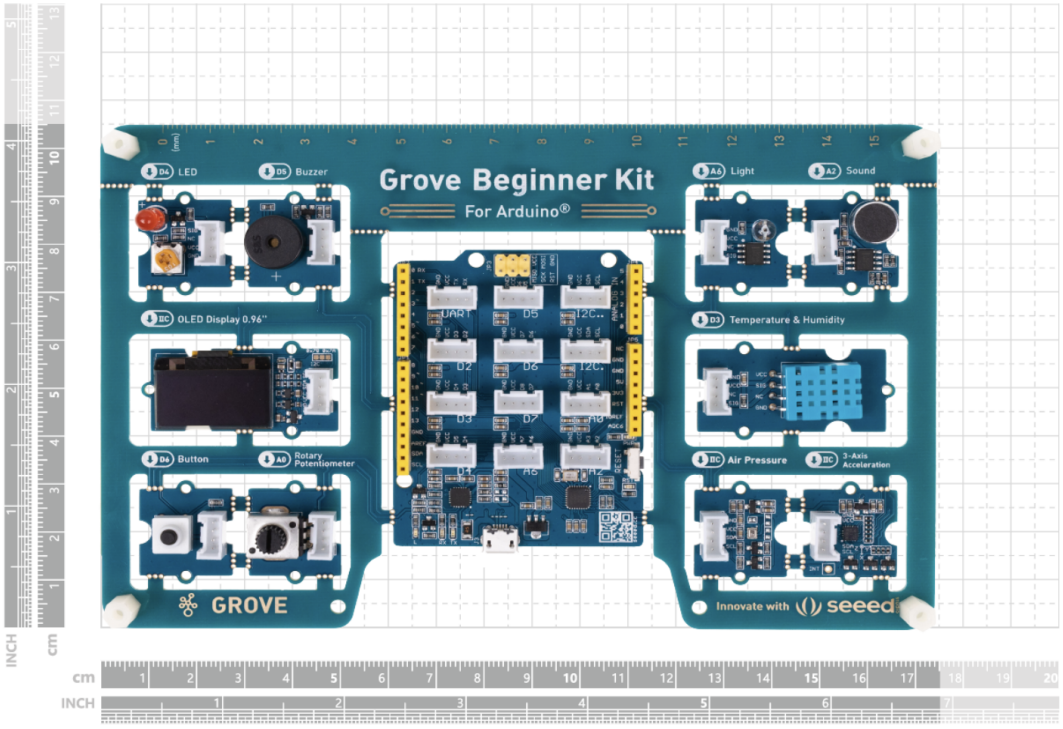

Figure 5.4: GBKA Dimensions [9]

malicious PC to perform the DoS attacks. Since we use a headless Raspberry Pi (i.e., it does not connect to the keyboard, mouse, and monitor), we enable USB to TTL connection on Raspberry Pi 4 (i.e. attacker PC). Therefore, we can communicate with Raspberry Pi 4 on our Macbook Pro. Moreover, as shown in Part H, we use jumpers to connect ATtiny85 and the universal PCB. In this research, ATtiny85 works as the

Table 5.1: Pin Assignments Between GBKA and NodeMCU

GBKA	**Logic Level Converter**		**NodeMCU**
Ground (GND)	GND (5V)	GND (3V3)	GND
5V	5V	3V3	3V3
RX pin	HV2	LV2	TX pin
TX pin	HV1	LV1	RX pin

external WDT.

Regarding the software part, we need to install the Raspberry Pi Operating System (OS) with desktop and recommended software on the Raspberry House. Figure 2.10 shows the operating system we use in our Raspberry House.

5.2 Enable USB to TTL Connection on Attacker PC

In this thesis, we use a USB to TTL serial cable, which is smaller and lighter than older serial cables. TTL stands for Transistor-Transistor Logic, a digital logic solution for processing and interpreting information. This serial connection is a traditional hardware connector that has been used for decades. Serial communication takes place over serial cables and is a linear form of data transmission, which means it sends data bit by bit through a communication port like a USB. It is a straightforward way of communication with a transmitter and receiver and is one of the most simple ways to communicate with a device.

In order to enable this USB to TTL functionality on the attacker computer (Raspberry Pi 4), we need to prepare the following hardware:

- A Raspberry Pi 4.
- A Micro-SD card with Linux distribution.
- A USB-to-TTL serial cable to connect Raspberry Pi 4 to our Macbook Pro.
- A Macbook Pro with an open USB port to communicate with the Pi through the cable

5.2.1 Install prerequisites on Raspberry Pi 4

Raspbian is the default operating system for the Raspberry Pi, and Raspbian has a Pi module called raspi-config which can help us easily enable the necessary settings we need. Therefore, we first need to install the raspi-config on our Raspberry Pi 4.

5.2.2 Enabling Connection on Raspberry Pi 4

After installing raspi-config on the Raspberry Pi 4, we will start connecting the UART cable to the Raspberry Pi 4's GPIO pins. Make sure that our Pi is not powered on and has the Micro-SD card inserted (inserting the SD card is optional). As shown in Figure 5.5, break out our serial cable and connect the following cables to the appropriate pins.

- We use the USB power supply in this thesis, so we ignore the power lead.
- Connect TXD cable to RXD on the Raspberry Pi 4.
- Connect RXD cable to TXD on the Raspberry Pi 4.
- Connect GND cable to GND on the Raspberry Pi 4.

5.2.3 Start The UART Connection

After we insert the micro-SD card into the Raspberry Pi 4, both our Macbook Pro and Raspberry Pi 4 are ready to communicate. To test it, we can open a terminal and enter the following command on the Macbook Pro terminal:

```
$ sudo screen /dev/tty.usbserial-ttyUSBport_number 115200
```

In addition, 115200 is the speed since more bits are to be transmitted when we enable the connection.

After executing the screen command, we can boot the Raspberry Pi 4. As shown in Figure 5.6, we successfully connect to the Raspberry Pi 4 on our Macbook Pro terminal using a UART connection.

At this point, we can quickly and easily power and log into the Raspberry Pi 4 from our Macbook Pro without the need for peripheral devices.

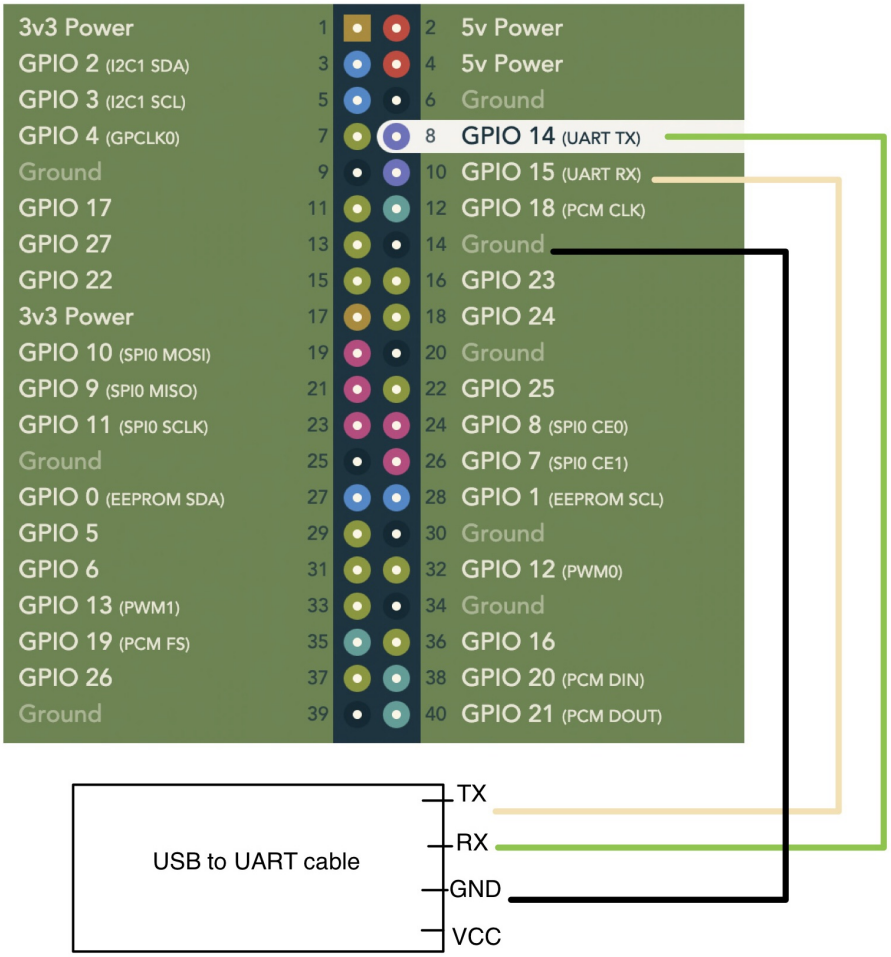

Figure 5.5: Headless Kali Linux UART Connection

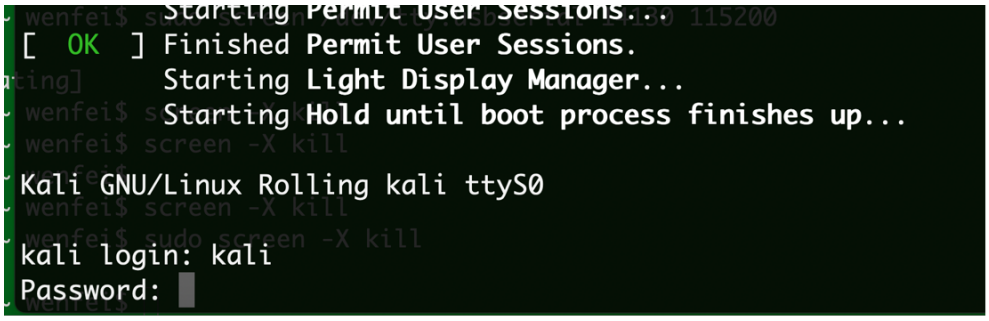

Figure 5.6: UART Connection Result

5.3 Raspberry House IDS and IPS Implementation

In this section, we will detail the implementation of IDS and IPS. Our IDS and IPS are based on shell scripts, Snort and WDT, and every shell script's running information

Table 5.2: Summary of the Systemd Services and Shell Scripts we used in thesis

Service Name (.service)	**Description**	**Run After (.service)**	**Shell Script Used By This Service (.sh)**	**Description**
mosquitto	Run mosquitto (MQTT broker) in the background.	- network (.target)	mosquitto-wrapper	Run mosquitto based on its configuration file.
enable-USBadapter	Enable USB WiFi Adapter as monitor mode to monitor the network traffic.	- network (.target)	enableAdapter	Enable the wireless interface of USB WiFi adapter as the monitor mode to monitor the network traffic.
MF_PUB	Run Python script which can detect the deauthentication (deauth) packets and publish to Topic A with local MQTT publisher (mosquitto_pub).	- network (.target) - mosquitto - MF_IDS - MF_IPS - MF_kotori-otokoAlert - MF_ emailAlert	- printDeauth.py - MF_Python	- printDeauth.py: Python script uses to detect the deauth packets. - MF_Python: a shell script to publish the result of printDeauth.py to Topic A using local MQTT publisher (mosquitto_pub)

will be stored in logs for future analysis. Moreover, to enable these shell scripts to run automatically when the Raspberry House is powered, we use the systemd service. Table 5.2 – 5.11 lists all the systemd services created in this thesis and the shell scripts used by the services.

As shown in the table, we need to use the mosquitto service for the data link layer, network layer and transport layer IDS and IPS. Since data link layer IDS and IPS are based on the Python script (**printDeauth.py**) and a shell script to detect and prevent the DoS attacks, and the network layer and transport layer are based on Snort to detect and prevent the DoS attack, we create two MQTT topics for these services. Table 5.12 classifies the services based on the MQTT topic name.

Table 5.3: Summary of the Systemd Services and Shell Scripts we used in thesis

Service Name (.service)	**Description**	**Run After (.service)**	**Shell Script Used By This Service (.sh)**	**Description**
MF_IDS	IDS for deauth attacks, which will blink the red LED on universal PCB after receive deauth packets.	- network (.target) - mosquitto	- MF_Blink	Use local MQTT subscriber (mosquitto_sub) to subscribe Topic A. If receive the deauth packets, then blink the red LED on universal PCB.
MF_IPS	IPS for deauth attacks, which will block the WiFi interface of Raspberry House.	- network (.target) - mosquitto	- disableWiFi - MF_IPS	- disableWiFi: block WiFi interface of Raspberry House, and unblock it after 15 seconds. - MF_IPS: use mosquitto_sub to subscribe Topic A. If the deauth packets received is greater than 5 (threshold), then start to run disableWiFi.sh.

Table 5.4: Summary of the Systemd Services and Shell Scripts we used in thesis (cont'd.)

Service Name (.service)	**Description**	**Run After (.service)**	**Shell Script Used By This Service (.sh)**	**Description**
MF_kotori-otokoAlert	Alert admin. through twitter message when get deauth attack.	- network (.target) - mosquitto	- kotoriotokoAlert - MF_kotoriotoko Alert	- kotoriotoko_ Alert: send a twitter alert to the administrator with the type of the DoS attack (deauth attack) and the timestamp. - MF_kotoriotoko_ Alert: use mosquitto_sub to subscribe Topic A. If receive any deauth packets, then run kotoriotokoAlert.sh.
MF_ emailAlert	Alert admin. through email when get deauth attack.	- network (.target) - mosquitto	- emailAlert - MF_emailAlert	- emailAlert: send an email alert to the administrator with the type of the attack (deauth attack) and the timestamp. - MF_emailAlert: use mosquitto_sub to subscribe Topic A. If receive any deauth packets, then run emailAlert.sh.

Table 5.5: Summary of the Systemd Services and Shell Scripts we used in thesis (cont'd.)

Service Name (.service)	**Description**	**Run After (.service)**	**Shell Script Used By This Service (.sh)**	**Description**
snort_PUB	Run Snort to catch the network layer attack (ICMP flood attack as an example in this thesis), and transport layer attack (SYN flood attack as an example in this thesis). Then publish the results to Topic B using local MQTT broker (mosquitto_pub).	- network (.target) - mosuiqtto - SYN_twitterAlert - SYN_emailAlert - SYN_blink - ICMP_twitterAlert - ICMP_emailAlert - ICMP_blink	snort_pub	Run Snort with its configuration in inline mode to capture any malicious packets defined in community ruleset and custom rules (SYN flood attack and ICMP flood attack). Then drop the packets defined in custom rules. Publish the results to Topic B using local MQTT publisher (mosquitto_pub).
SYN_blink	If receive any SYN flood DoS attack, then blink the blue LED on universal PCB.	- network (.target) - mosquitto	SYN_Blink	Use local MQTT subscriber (mosquitto_sub) to subscribe Topic B. If receive the SYN flood attack, then blink the blue LED on universal PCB.

Table 5.6: Summary of the Systemd Services and Shell Scripts we used in thesis (cont'd.)

Service Name (.service)	**Description**	**Run After (.service)**	**Shell Script Used By This Service (.sh)**	**Description**
SYN_ twitterAlert	Alert admin. through twitter messages when get SYN flood attack.	- network (.target) - mosquitto	- SYNtwitter - SYN_twitter-Alert	- SYNtwitter: send a twitter alert to the administrator with the type of the DoS attack (SYN flood attack) and the timestamp. - SYN_twitterAlert: use mosquitto_sub to subscribe Topic B. If receive any SYN flood DoS attack, then run SYNtwitter.sh
SYN_ emailAlert	Alert admin. through email when get SYN flood attack.	- network (.target) - mosquitto	- SYNemail - SYN_email-Alert	- SYNemail: send an email alert to the administrator with the type of the DoS attack (SYN flood attack) and the timestamp. - SYN_emailAlert: use mosquitto_sub to subscribe Topic B. If receive any SYN flood DoS attack, then run SYNemail.sh

Table 5.7: Summary of the Systemd Services and Shell Scripts we used in thesis (cont'd.)

Service Name (.service)	**Description**	**Run After (.service)**	**Shell Script Used By This Service (.sh)**	**Description**
ICMP_blink	If receive any ICMP flood DoS attack, then blink the green LED on universal PCB.	- network (.target) - mosquitto	- ICMP_Blink	Use local MQTT subscriber (mosquitto_sub) to subscribe Topic B. If receive the ICMP flood attack, then blink the green LED on universal PCB.
ICMP_ twitterAlert	Alert admin. through twitter message when get ICMP flood attack.	- network (.target) - mosquitto	- ICMPtwitter - ICMP_twitter Alert	- ICMPtwitter: send a twitter alert to the administrator with the type of the DoS attack (ICMP flood attack) and the timestamp. - ICMP_twitterAlert: use mosquitto_sub to subscribe Topic B. If receive any ICMP flood DoS attack, then run ICMPtwitter.sh

Table 5.8: Summary of the Systemd Services and Shell Scripts we used in thesis (cont'd.)

Service Name (.service)	Description	Run After (.service)	Shell Script Used By This Service (.sh)	Description
ICMP_ emailAlert	Alert admin. through email when get ICMP flood attack.	- network (.target) - mosquitto	- ICMPemail -ICMP_emailAlert	- ICMPemail: send an email alert to the administrator with the type of the DoS attack (ICMP flood attack) and the timestamp. - ICMP_emailAlert: use mosquitto_sub to subscribe Topic B. If receive any ICMP flood DoS attack, then run ICMPemail.sh

5.3.1 Data Link Layer IDS and IPS Implementation

For the IDS implementation of the data link layer, we created a shell script named **enableAdapter.sh** to make the USB WiFi adapter monitor network traffic, and used Python 2.7 to implement the deauthentication packet detection script mentioned in the design section. The USB WiFi adapter used in this experiment is the AWUS036NHA model of the Alfa brand. This model is an IEEE 802.11b/g/n wireless USB adapter and supports wireless data encryption using 64/128-bit WEP, WPA, WPA2, TKIP, and AES. In addition, this model uses 2.4GHz wavelength and supports MINO (multiple input multiple outputs) high-speed transmission TX data rate up to 150 Mbps.

enableAdapter.sh uses aircrack-ng to monitor network traffic. Aircrack-ng is a complete set of tools for assessing the security of WiFi networks. It can monitor the network traffic, implement attacks such as replay attacks, deauthentication, fake access points and others via packet injection, check WiFi cards and driver capabilities, and crack such as WEP and WPA PSK (WPA 1 and 2). All aircrack-ng tools are command lines, allowing for heavy scripting. Many GUIs take advantage of this feature. It mainly works on Linux but can also work on Windows, macOS, FreeBSD,

Table 5.9: Summary of the Systemd Services and Shell Scripts we used in thesis (cont'd.)

Service Name (.service)	**Description**	**Run After (.service)**	**Shell Script Used By This Service (.sh)**	**Description**
WDTgpio-Setup	Service to setup all GPIO pins on Raspberry House at startup.	N/A	WDTgpioSetup	- Set initial mode of all GPIO pins for attack notification to OUT mode. (i.e., red LED pin which indicate deauth attack, green LED pin which indicate ICMP flood attack, and blue LED pin which indicate SYN flood attack). - Set the initial mode of GPIO pin A which work to generate the heart beat pulse on Raspberry House and send to WDT built by ATtiny85 to OUTPUT HIGH. - Set the initial mode of the GPIO pin B which work as a switch between gentle WDT (when it set to HIGH) and delayed WDT (when it set to LOW) to OUTPUT HIGH. - Set the initial mode of the GPIO pin C which will receive the signal from ATtiny85 WDT (i.e. if it goes LOW level, then reboot Raspberry House) to INPUT_PULLUP.

Table 5.10: Summary of the Systemd Services and Shell Scripts we used in thesis (cont'd.)

Service Name (.service)	**Description**	**Run After (.service)**	**Shell Script Used By This Service (.sh)**	**Description**
kickWDT	Use GPIO pin A on Raspberry House to generate heart beat pulse and send to WDT built by ATtiny85.	- network (.target) - mosquitto	kickWDT	Start the heart beat pulse on Raspberry House. The purpose of this is that if the pulse is different than the defined one, which means the system does not run as normal, then the external WDT built by ATTiny85 will start to run based on the state of GPIO pin B and GPIO pin C.
watchdog	The built in (hardware) WDT on Raspberry House.	multi-user (.target)	N/A	N/A

Table 5.11: Summary of the Systemd Services and Shell Scripts we used in thesis (cont'd.)

Service Name (.service)	**Description**	**Run After (.service)**	**Shell Script Used By This Service (.sh)**	**Description**
WDT-notification	Run the external WDT built by ATtiny85, and send alert to Admin through twitter message and email.	- network (.target) - WDTgpio-Setup	WDTdelayed	Check the WDT GPIO pin C, if its state has been turned to LOW level (which means the gentle WDT start to run) for 6 seconds, then start the gentle WDT. Meanwhile, check the GPIO pin which control the delayed WDT (GPIO pin B), if its state has been turned to LOW also for 6 seconds, then start the delayed WDT, otherwise remain run gentle WDT. Before the external WDT start reboot Raspberry House, it will send an alert to the administrator through email and twitter message with the type of the external WDT (gentle or delayed WDT) and the timestamp.

Table 5.12: Classify Services Based on MQTT Topic

MQTT Topic	
wen/data1	**wen/data2**
MF_IDS.service	snort_PUB.service
MF_IPS.service	ICMP_blink.service
MF_PUB.service	SYN_blink.service
MF_kotoriotokoAlert.service	ICMP_emailAlert.service
MF_emailAlert.service	SYN_emailAlert.service
	ICMP_twitterAlert.service
	SYN_twitterAlert.service
mosquitto.service (used in both topics)	
*Notice: the service to setup GPIO pins of Raspberry House (WDTgpioSetup.service) and enable the USB WiFi adapter to monitor the network traffic (enableUSBadapter.service) do not need MQTT service to transfer data. In addition, the services for WDT (watchdog.service, WDTnotification.service, and kickWDT.service) also do not need to use MQTT service.	

OpenBSD, NetBSD, as well as Solaris and eComStation 2.

When first trying to use the airmon-ng tool to set the wireless network interface on the USB WiFi adapter to monitor mode, errors about the process interfering with the monitor mode occur. Therefore, we have to kill them before moving forward. Airmon-ng offers a command to kill them all easily:

```
$ airmon-ng check kill
```

Then, enable monitor mode on the wireless interface of the USB WiFi adapter by using the following command:

```
$ airmon-ng start wlan1
```

In this case, wlan1 is the wireless interface of the USB WiFi adapter. When we run this command on Raspberry House, it will return a confirmation, as shown in Figure 5.7, to show us that we successfully enable the wlan1 interface to work as monitor mode. As shown in Figure 5.8, the **Mode** of the wlan1 interface is **Monitor**.

In addition, we will save the running information of **enableAdapter.sh** to a log file for future research analysis. Therefore, the content of **enableAdapter.sh** is shown in Figure 5.9.

To be able to enable the script to run autonomously when the Raspberry House is

```
pi@wenpipi:~ $ sudo airmon-ng start wlan1

Found 5 processes that could cause trouble.
Kill them using 'airmon-ng check kill' before putting
the card in monitor mode, they will interfere by changing channels
and sometimes putting the interface back in managed mode

  PID Name
  306 avahi-daemon
  322 avahi-daemon
  345 wpa_supplicant
  384 dhcpcd
 1263 wpa_supplicant

PHY     Interface       Driver          Chipset

phy0    wlan0           brcmfmac        Broadcom 43430
phy1    wlan1           ath9k_htc       Qualcomm Atheros Communications AR9271 802.11n

                (mac80211 monitor mode already enabled for [phy1]wlan1 on [phy1]10)
```

Figure 5.7: Monitor Mode Setup Result

```
pi@wenpipi:~ $ iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.

wlan0     IEEE 802.11  Mode:Master  Tx-Power=31 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:on

wlan1     IEEE 802.11  Mode:Monitor  Frequency:2.457 GHz  Tx-Power=20 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:off
```

Figure 5.8: Monitor Mode Interface Chckeing

powered, we need the systemd service for this purpose. Figure 5.10 shows the service configuration for **enableAdapter.sh**. The Description is the purpose of the service, and we want to set our USB WiFi adapter to monitor mode after the network is set up. In addition, we set the **Type** of the service to **simple**, i.e. the service manager will consider the unit started immediately after the main service process has been forked off. It is expected that the process configured with **ExecStart** is the main process of the service. The **ExecStart** is the path of the program we want to run in our service. We want our service always to run when the Raspberry House is powered since we need to monitor the network traffic to capture the malicious packets. Furthermore, we set **WantedBy** equal to **multi-user.target**. **multi-user.target** usually defines a system state where all network services are up, and the system will accept logins. However, no local GUI is started, which is the typical default system state for a

```
#!/bin/sh

LOGFILE=/home/pi/log/$(basename $0).log
echo "" >> $LOGFILE
echo "[ $(/bin/date) ]" >> $LOGFILE
exec >> $LOGFILE 2>&1


/usr/sbin/airmon-ng check kill
/bin/sleep 1
/usr/sbin/airmon-ng start wlan1

/bin/sleep 3
```

Figure 5.9: Content of enableAdapter.sh

server system, possibly a rack-mounted headless system in a remote server room (i.e. it tells systemd that this service should be started as part of normal system start-up, whether or not a local GUI is active).

```
pi@wenpipi:/lib/systemd/system $ cat enableUSBadapter.service
[Unit]
Description=Enable USB network adapter to monitor wireless interface
After=network.target

[Service]
Type=simple
ExecStart=/home/pi/etc/enableAdapter.sh
restart=always


[Install]
WantedBy=multi-user.target
```

Figure 5.10: Content of enableAdapter.service

Then we create another shell script named **MF_Python.sh** that uses the local mosquitto_pub (i.e. MQTT broker tool) to publish the results of the Python script on a topic and then subscribe to the same topic in other shell scripts. When the Python script detects the deauthentication packets, we use msmtp and kotoriotoko

Table 5.13: The Functionality Provided By The Kotoriotoko [5]

Function	**Detail**
Posting	- Tweet - Retweet - Cancel a tweet - Like - Unlike
Tweets Viewing	- View Somebody's timeline - Search tweets by keywords
User Controlling	- Follow somebody - Unfollow somebody - List users who you following - List users who you are followed - View Somebody's info
Direct Message Managing	- Send - Receive - Delete - List
Other functions	- View trend list - Gather tweets in bulk continuously

to send alerts to users in the form of emails and Twitter, and blink the red LED on the universal PCB. The mosquitto client version we used in this thesis is version 1.4.10. The msmtp is an SMTP client that can be used to send mail from MUA (mail user agent). It transmits the message to the SMTP server in default mode, which delivers the final delivery (e.g. at free mail providers). By using configuration files, it can be easily configured to use different SMTP servers with different configurations, which makes it ideal for mobile clients. msmtp is free software and runs on a variety of platforms. The msmtp we use in this thesis is version 1.8.19, and we enable TLS encryption in its configuration file to ensure the security of the email. Kotoriotoko is a command set for operating Twitter [5], makes the users possible to operate Twitter on the command-line interface (CLI). Therefore, other applications on UNIX can operate Twitter more easily. Table 5.13 shows the functionality provided by the Kotoriotoko command. Our research uses it to send messages to Raspberry House's users to alert them to the type of DoS attack with a timestamp.

For the IPS implementation of the data link layer, after the Python script detects the deauthentication packets and publishes them to a specific topic, we create a shell

script named **MF_IPS** which uses mosquitto_sub to subscribe to that topic and check the received packets. If the number of deauthentication packets it received is greater than the threshold (we set the maximum deauthentication packets we accept to 5), we start the shell script to block the WiFi interface of Raspberry House. There are many ways to block the wireless interface. In our research, we use the rfkill tool to enable and disable the wireless device (i.e. the WLAN interface of Raspberry House). Rfkill is a subsystem in the Linux kernel that provides an interface through which radio transmitters in a computer system can be queried, activated, and deactivated. When transmitters are deactivated, they can be placed in a state where software can respond to them (soft block) or in which software cannot respond (hard block).

5.3.2 Network Layer and Transport Layer IDS and IPS Implementation

We use Snort inline mode and rules to detect and prevent the DoS attacks for network and transport layer IDS and IPS implementation. We use Snort custom rules (as shown in Figure 5.11) along with community rules to help detect and drop the malicious packets of network layer attacks (ICMP flood DoS attack as an example in this thesis) and transport layer attacks (TCP SYN flood DoS attack as an example in this thesis).

```
drop icmp any any -> $HOME_NET any (msg:"ICMP Flood Attack Rejected"; detection_filter\
:track by_dst, count 30, seconds 10; sid:1000001; rev:1;classtype:icmp-event;)
drop tcp any any -> $HOME_NET 80 (msg:"SYN Flood Attack Rejected"; detection_filter:tr\
ack by_dst, count 30, seconds 10; sid:1000002;rev:2;classtype:attempted-dos;)
```

Figure 5.11: Raspberry House Custom Snort Rules

We need to use Snort inline mode to trigger drop rules in our custom rules to prevent DoS attacks. In this thesis, enforcing Snort running inline (IPS) with DAQ AFPacket requires four significant configuration changes as follows:

- Configuring Snort policy to run inline (config option within Snort.conf).
- Configuring DAQ AFPacket to run inline (config option within Snort.conf, can be passed during runtime).
- Forcing Snort to run in inline mode with the `-Q` command line runtime argument.

- Modifying the rules to drop traffic on matches, i.e., changing "alert" to "drop" in the rules.

In addition, running Snort as an IPS with DAQ AFPacket does not require changing the iptables rules since Snort handles dropping the traffic. Moreover, when running Snort as an IPS with DAQ AFPacket, Snort itself bridges the interfaces used on the fly. No prior interface bridging/bonding is required.

5.3.3 System Security Level IDS and IPS Implementation

The system security level IDS and IPS implementation is based on the watchdog timer (WDT). We use the bash fork bomb as an example of system level attack in this thesis and design three WDTs (i.e., built-in WDT, gentle WDT and delayed WDT) to detect and prevent these attacks. The external WDT (i.e. gentle WDT and delayed WDT) built on ATtiny85 refers to the miniWDT on GitHub [89].

Following are the steps to enable Raspberry House built-in WDT.

- First, we need to activate watchdog hardware in Raspberry House.
- Then, we need to install the software side of this to communicate with the watchdog.
- We need to configure the watchdog to respond to events. We edit the `watchdog.conf` file in **etc** directory, and uncomment the following two lines.

  ```
  max-load-1 = 24
  watchdog-device = /dev/watchdog
  ```

  Since we want our built-in WDT to start 15 seconds after detecting the DoS attacks, we add one more line to the watchdog.config:

  ```
  watchdog-timeout = 15
  ```

  Note that the 15 seconds we set is the maximum value for hardware watchdog timeout.

- Then, we enable the watchdog service by using the following command.

  ```
  $ sudo systemctl enable watchdog
  $ sudo systemctl start watchdog
  ```

Table 5.14: Pin Assignments of the ATtiny85 and the Raspberry House

ATtiny85	**Raspberry House**	**Raspberry House GPIO Pin Initial Mode**
Pin0	GPIO A in Table 5.6	mode: OUTPUT logic level: HIGH
Pin1 (on board LED)	N/A	N/A
Pin2	GPIO B in Table 5.6	mode: OUTPUT logic level: HIGH
Pin3	RUN pin header	N/A (Run pin header is a direct connect to reset Raspberry House)
Pin4	GPIO C in Table 5.6	mode: INPUT_PULLUP
5V	5V	N/A
GND	GND	N/A

We check the status of the watchdog.service by using the following command.

```
$ sudo systemctl status watchdog
```

As shown in Figure 5.12, we successfully enable the built in WDT on Raspberry House

Figure 5.12: Built in WDT Service Status

Table 5.14 lists the pin assignments of the ATtiny85 (i.e. external WDT) and the Raspberry House initial GPIO modes and levels (states).

First, we start a heartbeat pulse on Raspberry House GPIO A using kickWDT.service. The ATTiny85 will detect if the received heartbeat pulse is the same as the defined interval (the defined heartbeat pulse is 1Hz, duty 30%, negative logic). If it is the same, it proves that the Raspberry House is working normally; if the heartbeat pulse received by ATTiny85 is different from the defined heartbeat pulse, it means that the Raspberry House system may be under a DoS attack which causes the system to

run slowly (such as the inode problem mentioned previously) or breaks it. When an anomaly is detected, i.e., the pulse is not sent from Raspberry House through the pin0 for more than T0 seconds, ATtiny85 will set pin4 to LOW and start gentle WDT, which will restart Raspberry House after T1 seconds. In order to restart Raspberry House, ATtiny85 will push its pin3, which is connected to the Raspberry House run pin header, to reset Raspberry House. In addition, it will check the state of pin2: if the state is still the same as the initial state (i.e. LOW), then ATiny85 will run as a gentle WDT (i.e.reboot Raspberry House after T1 seconds); otherwise, it will run as a delayed WDT (i.e., reboot Raspberry House after T2 seconds). Note that T2 is greater than T1 to meet our design purpose.

On the Raspberry House side, we design a WDTnotification.service. In this service, the Raspberry House will first check the state of GPIO C: if ATtiny85 sets it to LOW for more than 6 seconds, then it means the gentle WDT would run. Meanwhile, if the state of GPIO B is set to LOW for more than 6 seconds, it means Raspberry House would run delayed WDT instead of gentle WDT. If any WDT is active, Raspberry House will send an alert to the administrator with the WDT type and a timestamp through both Twitter messages and email.

5.3.4 Evaluation Scenario Implementation

The DoS attacks in this thesis for penetration testing using the Kali Linux is deauthentication attack, ICMP flood attack and SYN flood attack. We install the Kali Linux system on Raspberry Pi 4 and set it as the attacker's malicious system. We use the aircrack-ng suite to perform the deauthentication attack and use hping3 to perform ICMP/SYN flood attacks. For the system security level attack, we perform the bash fork bomb on Raspberry House to test the performance of WDTs.

Chapter 6

Experimental Results and Evaluation

This chapter presents experimental results to validate whether Raspberry House's IDS and IPS can effectively detect DoS attacks simulated by a malicious PC.

The results show that Raspberry House can detect deauthentication attacks via a Python script (Figure 6.1). This thesis uses the aircrack suite to simulate a deauthentication attack. We first start monitor mode on our malicious PC through the **airmon-ng** command, then use the **airodump-ng** command to scan WiFi networks to find our target, and finally perform the deauthentication attack by using the **aireplay-ng** command to send thousands of deauthenticate frames to keep anyone from reconnecting to the Raspberry House. The deauthentication attack overloaded the Raspberry House wireless interface performance approximately 1 minute after execution. We ended the deauthentication attack before the Raspberry House's wireless interface hung due to a request overflow. The average delay for Raspberry House to detect deauthentication packets is about 5.3 milliseconds. After the Raspberry House detects the deauthentication attack, the red LED on the universal PCB blinks to indicate the deauthentication attack and an alert is sent to the administrator via email and Twitter messages in real-time with the timestamp of the attack and type of the attack (as shown in Figure 6.2 and Figure 6.3). After detecting that the deauthentication packets exceed the threshold, the IPS service of the Raspberry House will start a cycle to block its WLAN interface for a preset interval for prevention purposes (as shown in Figure 6.4 and Figure 6.5). After confirming that there are no new deauthentication packets, the service will reopen the WLAN interface of the Raspberry House.

Raspberry House can detect ICMP flood attacks as well as SYN flood attacks through Snort community rules and Raspberry House custom rules. After the Raspberry House detects the above two attacks, the green LED on the universal PCB blinks to indicate the ICMP flood attack, whereas the blue LED on the universal

```
^Cpi@wenpipi:~/Desktop sudo python printDeauth.py
2021-11-16 23:05:36 1637118336227 Deauth_Detect_Against_Mac_Addr B8:27:EB:79:20:82
2021-11-16 23:05:36 1637118336239 Deauth_Detect_Against_Mac_Addr B8:27:EB:79:20:82
2021-11-16 23:05:36 1637118336292 Deauth_Detect_Against_Mac_Addr B8:27:EB:79:20:82
2021-11-16 23:05:36 1637118336295 Deauth_Detect_Against_Mac_Addr B8:27:EB:79:20:82
2021-11-16 23:05:36 1637118336342 Deauth_Detect_Against_Mac_Addr B8:27:EB:79:20:82
2021-11-16 23:05:36 1637118336363 Deauth_Detect_Against_Mac_Addr B8:27:EB:79:20:82
2021-11-16 23:05:36 1637118336632 Deauth_Detect_Against_Mac_Addr B8:27:EB:79:20:82
2021-11-16 23:05:36 1637118336667 Deauth_Detect_Against_Mac_Addr B8:27:EB:79:20:82
2021-11-16 23:05:36 1637118336745 Deauth_Detect_Against_Mac_Addr B8:27:EB:79:20:82
2021-11-16 23:05:36 1637118336793 Deauth_Detect_Against_Mac_Addr B8:27:EB:79:20:82
2021-11-16 23:05:36 1637118336797 Deauth_Detect_Against_Mac_Addr B8:27:EB:79:20:82
2021-11-16 23:05:36 1637118336815 Deauth_Detect_Against_Mac_Addr B8:27:EB:79:20:82
```

Figure 6.1: Deauthentication Attack Detection Results

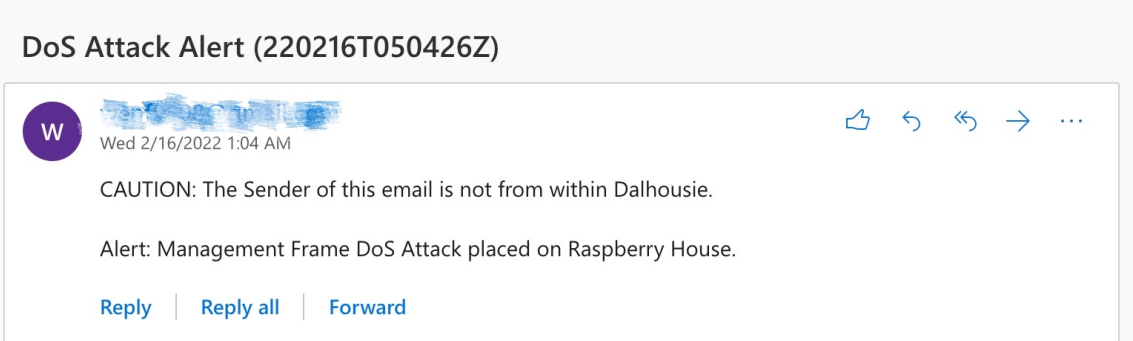

Figure 6.2: Deauthentication Attack Alert Through Email

Figure 6.3: Deauthentication Attack Alert Through Twitter Message

PCB blinks to indicate the SYN flood attack, and an alert is sent to the administrator through email and Twitter messages in real-time with the timestamp of the attack and type of the attack (as shown in Figure 6.6 - Figure 6.9).

Then we use Snort inline mode and Raspberry House custom rules to prevent ICMP flood attacks as well as SYN flood attacks. In this thesis, the ICMP flood attack and the SYN flood attack are all made using the hping3 command on the malicious PC. The results show that in the 5-minute test, the Raspberry House hardware performance is not affected by the ICMP flood attack or SYN flood attack, and drop rules can be triggered in Snort inline mode to drop malicious packets from the attack source (as shown in Figure 6.10 and Figure 6.11). Moreover, the delay of each ICMP

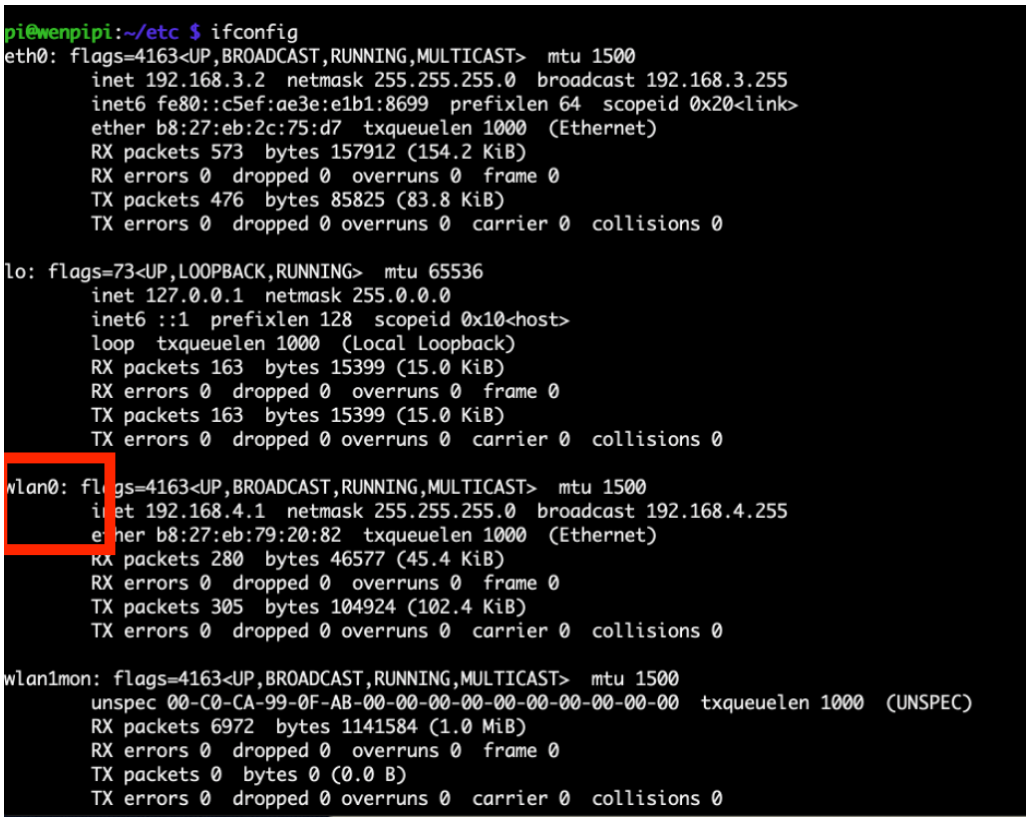

Figure 6.4: Normal Interfaces on Raspberry House

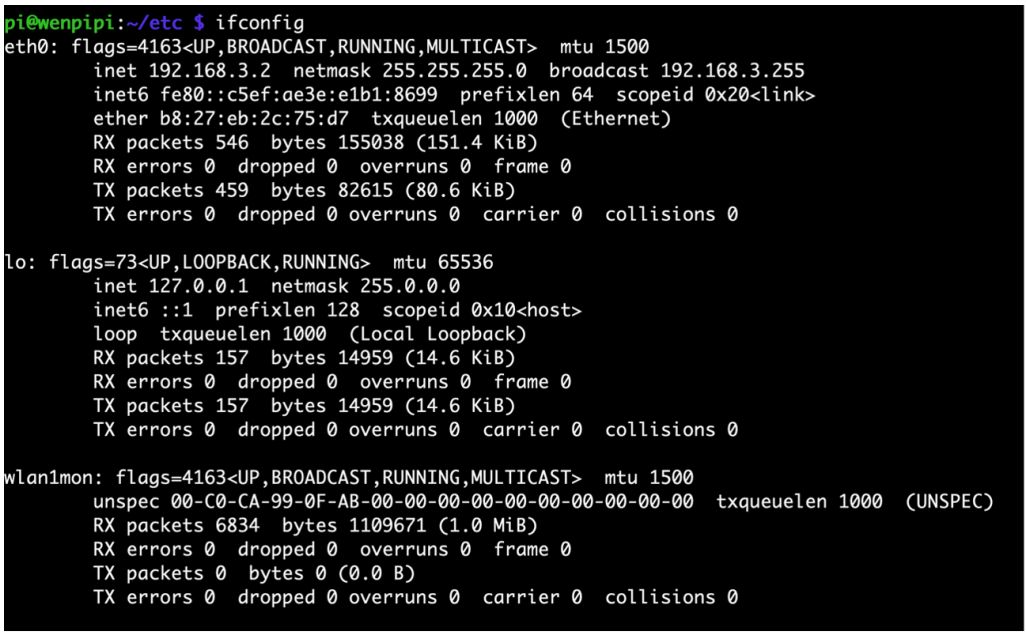

Figure 6.5: Raspberry House Interfaces Under Deauthentication Attack

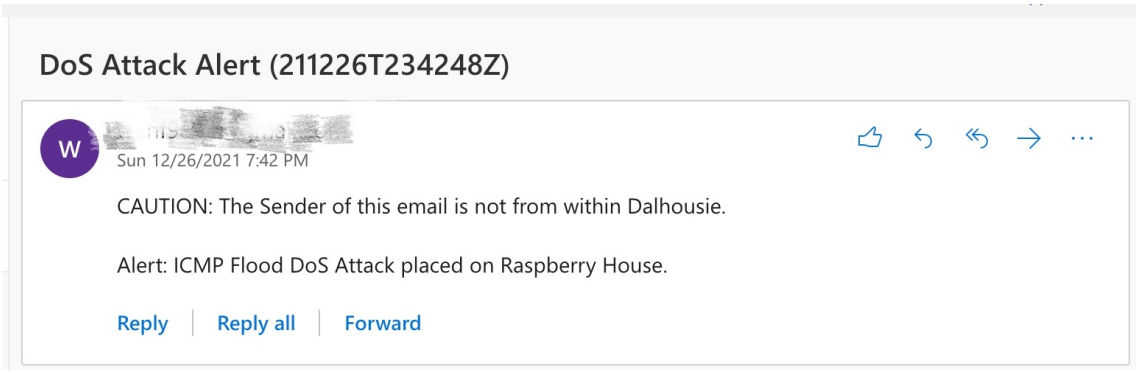

Figure 6.6: ICMP Flood DoS Attack Alert Through Email

Figure 6.7: ICMP Flood DoS Attack Alert Through Twitter Message

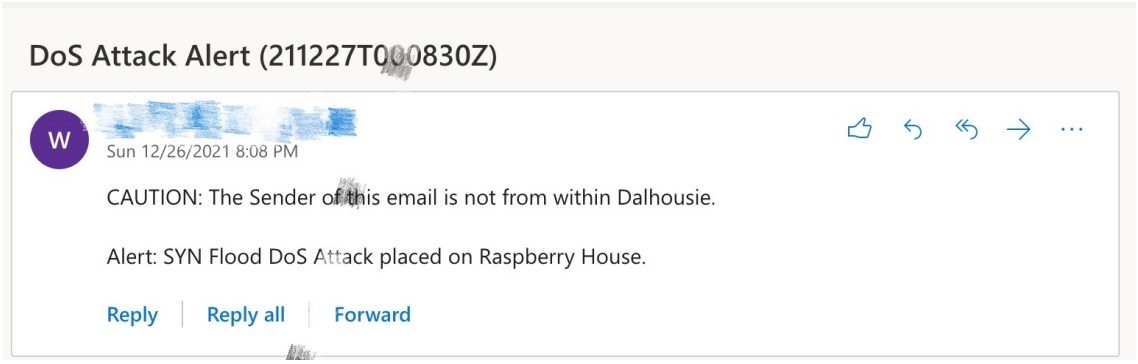

Figure 6.8: SYN Flood DoS Attack Alert Through Email

Figure 6.9: SYN Flood DoS Attack Alert Through Twitter Message

flood DoS attack to trigger the drop rule is about 9 milliseconds for another ICMP flood DoS attack, and the delay of each SYN flood DoS attack to trigger the drop rule is about 13 milliseconds for another SYN flood DoS attack.

IDS and IPS use Raspberry House's hardware WDT and external WDTs (i.e. gentle WDT and delayed WDT) to catch and prevent the bash fork bomb attack. The

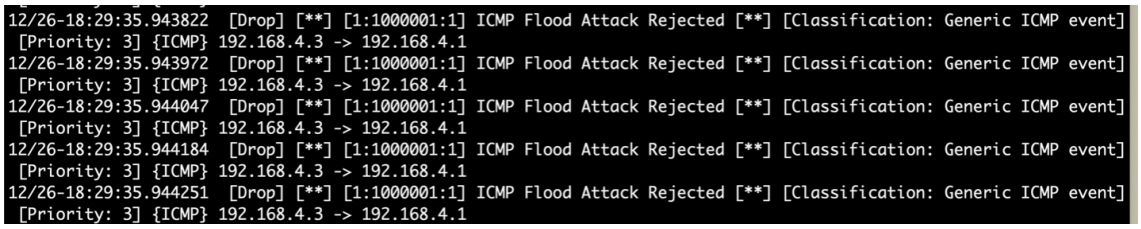

Figure 6.10: ICMP Flood DoS Attack IPS Drops Malicious Packets

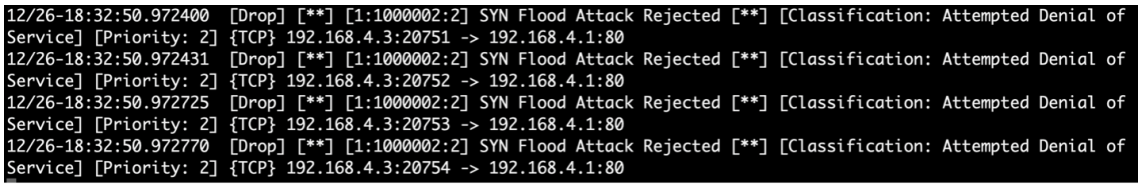

Figure 6.11: SYN Flood DoS Attack IPS Drops Malicious Packets

test results show that the bash fork bomb attack will immediately shut down or hang the Raspberry House. After executing the bash fork bomb attack, compared to the other two WDTs, the hardware watchdog can immediately detect the system anomaly of the Raspberry House and restart the Raspberry House after 15 seconds (as shown in Figure 6.12). To test the performance of the other two WDTs, we did a GPIO test on the Raspberry House (i.e., change the state of the GPIO pin corresponding to the target WDT to make it run). The results show that after confirming that the state of the target Raspberry House GPIO pin was changed for at least 6 seconds, either the gentle WDT or the delayed WDT was initiated depending on which GPIO pin state was changed.

Gentle WDT will wait for 6 seconds after detecting an anomaly of the Raspberry House system (such as an inode problem) to confirm that the state of the corresponding GPIO pin is not changed by accident. (delayed WDT uses the same method to detect its corresponding GPIO state). After confirming the Raspberry House GPIO pin status, if the gentle WDT is enabled, the Raspberry House will be restarted after 60 seconds (as shown in Figure 6.13); if the delayed WDT is enabled, the Raspberry House will be restarted after 120 seconds (as shown in Figure 6.14). In addition, an alert is sent to the administrator through email and Twitter messages in real-time with the timestamp and type of the external WDT (as shown in Figure 6.15 – Figure 6.18).

As mentioned in Chapter 3, many related studies only consider IoT IDS and ignore

```
pi@wenpipi:~ $ sudo bash -c ':(){ :|:& };:'
pi@wenpipi:~ $ Connection to wenpipi.local closed by remote host.
Connection to wenpipi.local closed.
WendeMacBook-Pro:Arduino15 wenfei$ ssh pi@wenpipi.local
pi@wenpipi.local's password:
Linux wenpipi 5.10.63-v7+ #1496 SMP Wed Dec 1 15:58:11 GMT 2021 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Dec 19 21:38:34 2021
pi@wenpipi:~ $ uptime
 21:38:56 up 0 min,  3 users,  load average: 1.02, 0.29, 0.10
pi@wenpipi:~ $
```

Figure 6.12: WDTs Test Result of Bash Fork Bomb Attack

```
pi@wenpipi:~/etc $ gpio mode 26 down
Broadcast message from root@wenpipi on pts/0 (Sat
2022-02-05 13:05:37 AST):

Reboot Raspberry House After 1 minute...
The system is going down for reboot at Sat
2022-02-05 13:06:37 AST!


Broadcast message from root@wenpipi on pts/0 (Sat
2022-02-05 13:06:38 AST):

Reboot Raspberry House After 1 minute...
The system is going down for reboot NOW!

Connection to wenpipi.local closed by remote host.
Connection to wenpipi.local closed.
WendeMacBook-Pro:~ wenfei$
```

Figure 6.13: Gentle WDT Result

```
pi@wenpipi:~/etc $ gpio write 22 0
pi@wenpipi:~/etc $ gpio -g read 6
0
pi@wenpipi:~/etc $
Broadcast message from root@wenpipi on pts/1 (Sat
2022-02-05 13:55:25 AST):

Reboot Raspberry House After 2 minute...
The system is going down for reboot at Sat 2022-02-05
13:57:25 AST!


Broadcast message from root@wenpipi on pts/1 (Sat
2022-02-05 13:56:25 AST):

Reboot Raspberry House After 2 minute...
The system is going down for reboot at Sat 2022-02-05
13:57:25 AST!


Broadcast message from root@wenpipi on pts/1 (Sat
2022-02-05 13:57:25 AST):

Reboot Raspberry House After 2 minute...
The system is going down for reboot NOW!

Connection to wenpipi.local closed by remote host.
Connection to wenpipi.local closed.
```

Figure 6.14: Delayed WDT Result

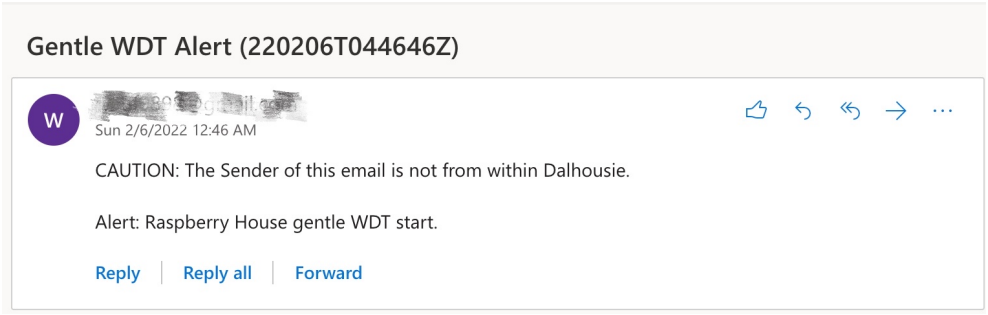

Figure 6.15: Gentle WDT Alert Through Email

Figure 6.16: Gentle WDT Alert Through Twitter Message

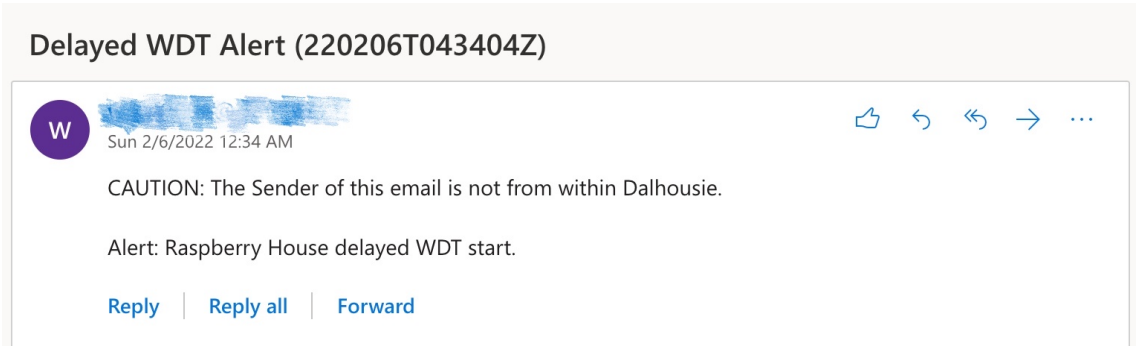

Figure 6.17: Delayed WDT Alert Through Email

Figure 6.18: Delayed WDT Alert Through Twitter Message

IPS. Therefore, this thesis focuses on IPS solutions based on different DoS attacks. Table 6.1 summarizes the DoS attacks covered in this thesis and the Raspberry House IPS approaches and shows our recommended IPS approach for different DoS attacks.

After detecting the deauthentication attack, we can use a shell script to block or unblock the Raspberry House WLAN interface or use WDTs to reboot Raspberry House after T seconds to prevent the deauthentication attack (T depends on the type of WDT we used). However, the best choice we recommend to prevent the deauthentication attack is to block the wireless interface of the Raspberry House for a certain time interval, since the deauthentication attack only affects the wireless connection between small IoT devices and Raspberry House, and the time interval for blocking the wireless interface of Raspberry House is shorter than restart time of WDTs, which will help Raspberry House to return to normal in a short time.

For both SYN flood DoS attack and ICMP flood DoS attack, we can use Snort inline mode along with custom Snort rules to drop the malicious packets or use WDTs

Table 6.1: Raspberry House IPS Recommendation

	Snort custom rules & inline mode	**Block & unblock wlan interface of Raspberry House**	**Raspberry House built in (hardware) WDT**	**Raspberry House gentle WDT**	**Raspberry House delayed WDT**
Deauth-entication Attack (Manag-ement Frame Attack)		C best choice	C		
ICMP Flood Attack	C best choice		C		
SYN Flood Attack	C best choice		C		
Bash Fork Bomb Attack			C best choice	C	C
Other continuous flood type DoS attacks			C	C	C best choice
Other 'gentle' DoS attack such as inode problem				C best choice	C
	C: covered				

to reboot Raspberry House after T seconds (T depends on the type of WDT we used) to prevent SYN flood DoS attack and ICMP flood DoS attack. The best choice we recommend to prevent SYN flood DoS attack and ICMP flood DoS attack is to run Snort in inline mode and use custom Snort rules to drop the malicious packets defined in the rules. The reason for choosing the Snort approach as the best IPS for both attacks is that Snort drops malicious packets directly from the attacker's source without affecting the Raspberry House's hardware performance. Therefore, We do not need to waste time using WDTs to reboot the Raspberry House.

For system security level DoS attacks like the bash fork bomb, we can use all types of WDTs as IPS. However, the best choice we recommend is to use the built-in WDT, as it responds the fastest when all three WDTs are enabled simultaneously (as shown in Figure 6.12). We recommend using the gentle WDT when a DoS attack looks like the **normal** way (such as consuming all the free inodes on the disk by creating a large number of temporary empty folders) and is difficult to detect using the built-in WDT since even if the heartbeat pulse sent by the Raspberry House has slight deviation, the gentle WDT also kicks in immediately. Finally, for the DoS attacks that will perform for a long time, the delayed WDT is the best choice because it has the longest time interval before resetting the Raspberry House compared to the other two WDTs.

Since this thesis aims to design a portable and low-cost IoT gateway, we need to ensure that the proposed Raspberry House is affordable for most IoT researchers or engineers. The total cost to build the system is $188CA, including Raspberry Pi 3B+ Extreme Kit $99CA, universal PCB $26CA, ATtiny85 board $3CA, and USB WiFi adapter $60CA. Note that all hardware we used in this thesis was bought from Amazon Canada.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis aims to create a secure and efficient IoT gateway for those IoT researchers and engineers to enable them to work on IoT devices in a secure network environment. The proposed Raspberry House handles DoS attacks on IoT devices, which is also efficient for small IoT devices with limited performance, such as GBKA. In our research, we use Raspberry Pi 3B+ to build the proposed gateway (i.e., Raspberry House) and investigate the security of Raspberry House. In addition, the Raspberry House security parameters were tested by simulating the most commonly performed DoS attacks against the Raspberry House network to show whether the Raspberry House IDS and IPS can detect and prevent these attacks in real-time. Furthermore, we summarize the DoS attacks proposed in this thesis and the feasible IPS and provide readers with our suggested optimal solutions for different DoS attacks. The results show that Raspberry House IDS and IPS can detect, alert and prevent DoS attacks proposed in this thesis in real-time, including deauthentication attack, SYN flood DoS attack, ICMP flood DoS attack and bash fork bomb attack. Moreover, Raspberry House is very economical, costing only $188CA.

7.2 Discussion

Given the current state of the Raspberry House, it has the potential to expand further.

The researchers can use other kinds of Linux based single boards instead of Raspberry Pi to build the Raspberry House, since currently there are multiple single boards which can work the same as Raspberry Pi, such as banana pi. In addition, since our IDS and IPS and based on shell script, which is portable and sustainable compared to C++ and other programming approaches. The researchers can easily move our IDS and IPS approach to other Linux based single boards. The reason we choose to

use Raspberry Pi in our thesis is that Raspberry Pi is one of the famous boards and is easy to buy.

Moreover, it is possible that the attacker only attacks the IoT devices such as the NodeMCU board used in our thesis. However, the NodeMCU board cannot send alerts to the administrator as this would be too heavy for it to process. Therefore, the administrator cannot take any action when under the DoS attack.

Furthermore, even if the WiFi connection between our IoT devices and Raspberry House is able to be attacked by attackers (i.e., the attacker can perform DoS attacks on the IoT devices), our Raspberry House system can detect the anomaly and cut off the connection between IoT devices and Raspberry House. For example, Raspberry House can use WDTs to reboot itself after an interval Therefore, the attacker cannot find the target anymore.

7.3 Future Work

In future work, we plan to duplicate Raspberry House in various research environments to test how well Raspberry House can work in different network environments. In addition, we plan to implement more lightweight IoT gateway applications dedicated to low-performance IoT devices to further improve the performance of Raspberry House. Furthermore, we plan to design other IDS and IPS for Raspberry House to prevent other common types of IoT attacks, such as Man-in-the-Middle attacks, making our Raspberry House more robust.

Bibliography

[1] P. Kirkbride, in *Basic Linux Terminal Tips and Tricks*. Apress, 2020, pp. XXV, 361.

[2] B. Caswell and J. Beale, *Snort 2.1 intrusion detection*. Elsevier, 2004.

[3] R. U. Rehman, *Intrusion detection systems with Snort: advanced IDS techniques using Snort, Apache, MySQL, PHP, and ACID*. Prentice Hall Professional, 2003.

[4] http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node31.html, accessed: 2022-3-14.

[5] "kotoriotoko: KOTORIOTOKO (little bird man) – extremely compatible and sustainable twitter application written in shell script," https://github.com/ShellShoccar-jpn/kotoriotoko.

[6] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE communications surveys & tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[7] S. N. Swamy and S. R. Kota, "An empirical study on system level aspects of internet of things (iot)," *IEEE Access*, vol. 8, pp. 188 082–188 134, 2020.

[8] W. Iqbal, H. Abbas, M. Daneshmand, B. Rauf, and Y. A. Bangash, "An in-depth analysis of iot security requirements, challenges, and their countermeasures via software-defined security," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10 250–10 276, 2020.

[9] "Grove beginner kit for arduino - all-in-one arduino compatible board with 10 sensors and 12 projects with free course," https://www.seeedstudio.com/Grove-Beginner-Kit-for-Arduino-p-4549.html, Jan. 2022, accessed: 2022-3-14.

[10] "IoT connections to grow 140% to hit 50 billion by 2022, as edge computing accelerates ROI," https://www.juniperresearch.com/press/iot-connections-to-grow-140pc-to-50-billion-2022?, accessed: 2022-3-13.

[11] "By 2030, each person will own 15 connected devices," https://www.toolbox.com/marketing/iot-in-marketing/articles/by-2030-each-person-will-own-15-connected-devices-heres-what-that-means-for-your-business-and-content/, accessed: 2022-3-13.

[12] C. Sobin, "A survey on architecture, protocols and challenges in iot," *Wireless Personal Communications*, vol. 112, no. 3, pp. 1383–1429, 2020.

[13] H. Mrabet, S. Belguith, A. Alhomoud, and A. Jemai, "A survey of iot security based on a layered architecture of sensing and data analysis," *Sensors*, vol. 20, no. 13, p. 3625, 2020.

[14] V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal, and B. Sikdar, "A survey on iot security: application areas, security threats, and solution architectures," *IEEE Access*, vol. 7, pp. 82 721–82 743, 2019.

[15] A. Triantafyllou, P. Sarigiannidis, and T. D. Lagkas, "Network protocols, schemes, and mechanisms for internet of things (iot): Features, open challenges, and trends," *Wireless communications and mobile computing*, vol. 2018, 2018.

[16] B. K. Mohanta, D. Jena, U. Satapathy, and S. Patnaik, "Survey on iot security: challenges and solution using machine learning, artificial intelligence and blockchain technology," *Internet of Things*, vol. 11, p. 100227, 2020.

[17] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying iot security: An exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2702–2733, 2019.

[18] M. A. Obaidat, S. Obeidat, J. Holst, A. Al Hayajneh, and J. Brown, "A comprehensive and systematic survey on the internet of things: security and privacy challenges, security frameworks, enabling technologies, threats, vulnerabilities and countermeasures," *Computers*, vol. 9, no. 2, p. 44, 2020.

[19] H. Wu, H. Han, X. Wang, and S. Sun, "Research on artificial intelligence enhancing internet of things security: A survey," *Ieee Access*, vol. 8, pp. 153 826–153 848, 2020.

[20] J. Fruhlinger, "The mirai botnet explained: How IoT devices almost brought down the internet," https://www.csoonline.com/article/3258748/the-mirai-botnet-explained-how-teen-scammers-and-cctv-cameras-almost-brought-down-the-internet.html, Mar. 2018, accessed: 2022-3-13.

[21] Z. A. Baig, S. Sanguanpong, S. N. Firdous, T. G. Nguyen, C. So-In *et al.*, "Averaged dependence estimators for dos attack detection in iot networks," *Future Generation Computer Systems*, vol. 102, pp. 198–209, 2020.

[22] "2021 web, mobile, IoT cybersecurity statistics," https://www.vaadata.com/blog/2021-web-mobile-iot-cybersecurity-statistics-strengthen-your-security-with-pentest/, Sep. 2021, accessed: 2022-3-14.

[23] V. Tsiatsis, S. Karnouskos, J. Holler, D. Boyle, and C. Mulligan, *Internet of Things: technologies and applications for a new age of intelligence*. Academic Press, 2018.

[24] S. Kajwadkar and V. K. Jain, "A novel algorithm for dos and ddos attack detection in internet of things," in *2018 Conference on Information and Communication Technology (CICT)*. IEEE, 2018, pp. 1–4.

[25] A. Mubarakali, K. Srinivasan, R. Mukhalid, S. C. Jaganathan, and N. Marina, "Security challenges in internet of things: Distributed denial of service attack detection using support vector machine-based expert systems," *Computational Intelligence*, vol. 36, no. 4, pp. 1580–1592, 2020.

[26] M. Anirudh, S. A. Thileeban, and D. J. Nallathambi, "Use of honeypots for mitigating dos attacks targeted on iot networks," in *2017 International Conference on Computer, Communication and Signal Processing (ICCCSP)*, 2017, pp. 1–4.

[27] B.-C. Chifor, I. Bica, and V.-V. Patriciu, "Mitigating dos attacks in publish-subscribe iot networks," in *2017 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, 2017, pp. 1–6.

[28] M. A. A. Mamun, "Real-time integration of iot sensor and iota tangle for securing iot infrastructure," 2022.

[29] A. Arora, "Preventing wireless deauthentication attacks over 802.11 networks," *arXiv preprint arXiv:1901.07301*, 2018.

[30] M. A. C. Aung and K. P. Thant, "Detection and mitigation of wireless link layer attacks," in *2017 IEEE 15th international conference on software engineering research, management and applications (SERA)*. IEEE, 2017, pp. 173–178.

[31] B. Sliwa, N. Piatkowski, and C. Wietfeld, "Limits: Lightweight machine learning for iot systems with resource limitations," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 2020, pp. 1–7.

[32] T. Zachariah, N. Klugman, B. Campbell, J. Adkins, N. Jackson, and P. Dutta, "The internet of things has a gateway problem," in *Proceedings of the 16th international workshop on mobile computing systems and applications*, 2015, pp. 27–32.

[33] W. Castellanos, J. Macias, H. Pinilla, and J. D. Alvarado, "Internet of things: a multiprotocol gateway as solution of the interoperability problem," *arXiv preprint arXiv:2108.00098*, 2021.

[34] A. Glória, F. Cercas, and N. Souto, "Design and implementation of an iot gateway to create smart environments," *Procedia Computer Science*, vol. 109, pp. 568–575, 2017, 8th International Conference on Ambient Systems, Networks and Technologies, ANT-2017 and the 7th International Conference on Sustainable Energy Information Technology, SEIT 2017, 16-19 May 2017, Madeira, Portugal. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050917310128

[35] L. Boyanov, V. Kisimov, and Y. Christov, "Evaluating iot reference architecture," in *2020 International Conference Automatics and Informatics (ICAI)*. IEEE, 2020, pp. 1–5.

[36] A.-E. Bouaouad, A. Cherradi, S. Assoul, and N. Souissi, "The key layers of iot architecture," in *2020 5th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech)*, 2020, pp. 1–4.

[37] A. Wolff, A. Seffah, G. Kortuem, and J. Van Der Linden, "Designing for effective interactions with data in the internet of things," in *Proceedings of the 2018 ACM Conference Companion Publication on Designing Interactive Systems*, 2018, pp. 415–418.

[38] D. Sehrawat and N. S. Gill, "Smart sensors: Analysis of different types of iot sensors," in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*. IEEE, 2019, pp. 523–528.

[39] K. Kaur, "A survey on internet of things–architecture, applications, and future trends," in *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*. IEEE, 2018, pp. 581–583.

[40] S. Smys, "A survey on internet of things (iot) based smart systems," *Journal of ISMAC*, vol. 2, no. 04, pp. 181–189, 2020.

[41] A. Bragarenco, "Sensor-actuator software component stack for industrial internet of things applications," in *2020 24th International Conference on System Theory, Control and Computing (ICSTCC)*. IEEE, 2020, pp. 540–545.

[42] H. Chi, T. Aderibigbe, and B. C. Granville, "A framework for iot data acquisition and forensics analysis," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 5142–5146.

[43] I. Boukhennoufa, A. Amira, F. Bensaali, and S. S. Esfahani, "A novel gateway-based solution for remote elderly monitoring," *Journal of Biomedical Informatics*, vol. 109, p. 103521, 2020.

[44] Z. Zhao, P. Lin, L. Shen, M. Zhang, and G. Q. Huang, "Iot edge computing-enabled collaborative tracking system for manufacturing resources in industrial park," *Advanced Engineering Informatics*, vol. 43, p. 101044, 2020.

[45] M. Alrowaily and Z. Lu, "Secure edge computing in iot systems: review and case studies," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 440–444.

[46] Z. Lv, L. Qiao, and S. Verma, "Ai-enabled iot-edge data analytics for connected living," *ACM Transactions on Internet Technology*, vol. 21, no. 4, pp. 1–20, 2021.

[47] N. Almolhis, A. M. Alashjaee, S. Duraibi, F. Alqahtani, and A. N. Moussa, "The security issues in iot-cloud: a review," in *2020 16th IEEE International Colloquium on Signal Processing & Its Applications (CSPA)*. IEEE, 2020, pp. 191–196.

[48] M. Goudarzi, H. Wu, M. Palaniswami, and R. Buyya, "An application placement technique for concurrent iot applications in edge and fog computing environments," *IEEE Transactions on Mobile Computing*, vol. 20, no. 4, pp. 1298–1311, 2020.

[49] G. Kumar *et al.*, "Understanding denial of service (dos) attacks using osi reference model," *International Journal of Education and Science Research*, vol. 1, no. 5, 2014.

[50] H. S. Obaid and E. H. Abeed, "Dos and ddos attacks at osi layers," *International Journal of Multidisciplinary Research and Publications*, p. 9, 2020.

[51] Z. Gavric and D. Simic, "Overview of DOS attacks on wireless sensor networks and experimental results for simulation of interference attacks," *IngenierÃa e InvestigaciÃ*, vol. 38, pp. 130 – 138, 04 2018. [Online]. Available: http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0120-56092018000100130&nrm=iso

[52] Y. Kristiyanto and E. Ernastuti, "Analysis of deauthentication attack on ieee 802.11 connectivity based on iot technology using external penetration test," *CommIT Journal*, vol. 14, no. 1, 2020.

[53] Y. Zhang and S. Sampalli, "Client-based intrusion prevention system for 802.11 wireless lans," in *2010 IEEE 6th International Conference on Wireless and Mobile Computing, Networking and Communications*, 2010, pp. 100–107.

[54] C. V. and S. P, "Icmp flood attacks: A vulnerability analysis," in *Cyber Security. Advances in Intelligent Systems and Computing,*, vol. 729. Springer, Singapore, 2018.

[55] N. Gupta, A. Jain, P. Saini, and V. Gupta, "Ddos attack algorithm using icmp flood," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2016, pp. 4082–4084.

[56] H. Harshita, "Detection and prevention of icmp flood ddos attack," *International Journal of New Technology and Research*, vol. 3, no. 3, 3 2017.

[57] D. Kshirsagar, S. Sawant, A. Rathod, and S. Wathore, "Cpu load analysis minimization for tcp syn flood detection," *Procedia Computer Science*, vol. 85, pp. 626–633, 2016, international Conference on Computational Modelling and Security (CMS 2016). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050916305786

[58] I. P. A. E. Pratama, "Tcp syn flood (dos) attack prevention using spi method on csf: A poc," *Bulletin of Computer Science and Electrical Engineering*, vol. 1, no. 2, p. 63–72, Aug. 2020. [Online]. Available: http://www.bcsee.org/index.php/bcsee/article/view/tcp-syn-flood-dos-attack-prevention-using-spi-method-on-csf-a-po

[59] K. Shah and K. Patel, "Security against fork bomb attack in linux based systems," *International Journal of Research in Advent Technology*, vol. 7, pp. 125–128, 04 2019.

[60] G. Nakagawa and S. Oikawa, "Fork bomb attack mitigation by process resource quarantine," in *2016 Fourth International Symposium on Computing and Networking (CANDAR)*, 2016, pp. 691–695.

[61] N. S. Yamanoor and S. Yamanoor, "High quality, low cost education with the raspberry pi," in *2017 IEEE Global Humanitarian Technology Conference (GHTC)*, 2017, pp. 1–5.

[62] https://www.amazon.ca/RS-Components-Raspberry-Model-Motherboard/dp/B07BFH96M3, accessed: 2022-3-14.

[63] W. Fei, H. Ohno, and S. Sampalli, "Design and implementation of raspberry house: An iot security framework," in *2020 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS)*, 2021, pp. 1–7.

[64] S. L. Jurj, R. Rotar, F. Opritoiu, and M. Vladutiu, "White-box testing strategy for a solar tracking device using nodemcu lua esp8266 wi-fi network development board module," in *2018 IEEE 24th International Symposium for Design and Technology in Electronic Packaging (SIITME)*, 2018, pp. 53–60.

[65] D. Soni and A. Makwana, "A survey on mqtt: A protocol of internet of things(iot)," 04 2017.

[66] T. Olsson, *Arduino wearables*. Springer, 2012.

[67] F. Björklund and N. Landin, "Board and chip diversity in deep learning side-channel attacks: On attiny85 implementations featuring encryption and communication," 2021.

[68] A. Ullah, S. I. Ullah, and A. Salam, "Internal dos attack detection and prevention in fog computing," in *2021 International Conference on Information Technology (ICIT)*. IEEE, 2021, pp. 763–768.

[69] V. Simadiputra and N. Surantha, "Rasefiberry: Secure and efficient raspberry-pi based gateway for smarthome iot architecture," *Bulletin of Electrical Engineering and Informatics*, vol. 10, no. 2, pp. 1035–1045, 2021.

[70] X.-H. Nguyen, X.-D. Nguyen, H.-H. Huynh, and K.-H. Le, "Realguard: A lightweight network intrusion detection system for iot gateways," *Sensors*, vol. 22, no. 2, p. 432, 2022.

[71] S. Javanmardi, M. Shojafar, R. Mohammadi, A. Nazari, V. Persico, and A. Pescapè, "Fupe: A security driven task scheduling approach for sdn-based iot–fog networks," *Journal of Information Security and Applications*, vol. 60, p. 102853, 2021.

[72] P. Binu, D. Mohan, and E. S. Haridas, "An sdn-based prototype for dynamic detection and mitigation of dos attacks in iot," in *2021 Third International Conference on Inventive Research in Computing Applications (ICIRCA)*. IEEE, 2021, pp. 5–10.

[73] W. B. W. Mariam and Y. Negash, "Performance evaluation of machine learning algorithms for detection of syn flood attack," in *2021 IEEE AFRICON*. IEEE, 2021, pp. 1–6.

[74] G. Shang, Y. Chen, C. Zuo, and Y. Zhu, "Design and implementation of a smart iot gateway," in *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, 2013, pp. 720–723.

[75] T. Zachariah, N. Klugman, B. Campbell, J. Adkins, N. Jackson, and P. Dutta, "The internet of things has a gateway problem," in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 27–32. [Online]. Available: https://doi.org/10.1145/2699343.2699344

[76] Y.-W. Chen, J.-P. Sheu, Y.-C. Kuo, and N. Van Cuong, "Design and implementation of iot ddos attacks detection system based on machine learning," in *2020 European Conference on Networks and Communications (EuCNC)*. IEEE, 2020, pp. 122–127.

[77] B. Gupta, P. Chaudhary, X. Chang, and N. Nedjah, "Smart defense against distributed denial of service attack in iot networks using supervised learning classifiers," *Computers & Electrical Engineering*, vol. 98, p. 107726, 2022.

[78] M. A. Ferrag, L. Shu, H. Djallel, and K.-K. R. Choo, "Deep learning-based intrusion detection for distributed denial of service attack in agriculture 4.0," *Electronics*, vol. 10, no. 11, p. 1257, 2021.

[79] L. Nie, Y. Wu, X. Wang, L. Guo, G. Wang, X. Gao, and S. Li, "Intrusion detection for secure social internet of things based on collaborative edge computing: A generative adversarial network-based approach," *IEEE Transactions on Computational Social Systems*, 2021.

[80] R. SaiSindhuTheja and G. K. Shyam, "An efficient metaheuristic algorithm based feature selection and recurrent neural network for dos attack detection in cloud computing environment," *Applied Soft Computing*, vol. 100, p. 106997, 2021.

[81] S. Velliangiri, P. Karthikeyan, and V. Vinoth Kumar, "Detection of distributed denial of service attack in cloud computing using the optimization-based deep networks," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 33, no. 3, pp. 405–424, 2021.

[82] B. F. L. M. Sousa, Z. Abdelouahab, D. C. P. a. Lopes, N. C. Soeiro, and W. F. Ribeiro, "An intrusion detection system for denial of service attack detection in internet of things," in *Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing*, ser. ICC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3018896.3018962

[83] A. Mihoub, O. B. Fredj, O. Cheikhrouhou, A. Derhab, and M. Krichen, "Denial of service attack detection and mitigation for internet of things using looking-back-enabled machine learning techniques," *Computers & Electrical Engineering*, vol. 98, p. 107716, 2022.

[84] R. Saxena and S. Dey, "Ddos attack prevention using collaborative approach for cloud computing," *Cluster Computing*, vol. 23, no. 2, pp. 1329–1344, 2020.

[85] S. Kumar and R. Amin, "Mitigating distributed denial of service attack: Blockchain and s oftware-defined networking based approach, network model with future research challenges," *Security and Privacy*, vol. 4, no. 4, p. e163, 2021.

[86] V. BIELIAVIN, "The use of watchdog timer for restoration of microprocessor systems with enhanced fault tolerance to influence of electrostatic discharges," 2021.

[87] D. R. SenthamilSelvan, D. V. Mahalakshmi, D. S. Vijayaragavan, D. S. Arulselvi, and D. Jasmin, "A novel watchdog timer for real-time intensive applications," 2021.

[88] J. Henry and View my complete profile, "CCIE wireless," http://wirelessccie.blogspot.com/2016/01/80211w-aka-pmf.html, accessed: 2022-3-14.

[89] H. Ohno, "Arduino/sketch_20220201_miniWDT at main · hohno-46466/rZone–RaspberryGate–WDT," https://github.com/hohno-46466/rZone--RaspberryGate--WDT/tree/main/Arduino/sketch_20220201_miniWDT.