KWAME NKRUMAH UNIVERSITY OF SCIENCE AND TECHNOLOGY

COLLEGE OF ENGINEERING

A Security Shield for Internet of Things (IoT) Devices

Justice Owusu Agyemang

(BSc. Telecommunication Engineering)

A THESIS SUBMITTED TO THE DEPARTMENT OF ELECTRICAL/ELECTRONIC ENGINEERING, KWAME NKRUMAH UNIVERSITY OF SCIENCE AND TECHNOLOGY, IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF MPhil. TELECOMMUNICATION ENGINEERING

June 2019

Declaration

I hereby declare that this submission is my own work towards the award of the MPhil degree and that, to the best of our knowledge, it contains no material previously published by another person or material which has been accepted for the award of any other degree of the university, except where due acknowledgment has been made in the thesis.

KNUST

Justice Owusu Agyemang	.....	.....
Student (20532486)	Signature	Date

Certified by:

Ing. Dr. Jerry John Kponyo	.....	.....
Supervisor	Signature	Date

Certified by:

Ing. Dr. Abdul-Rahman Ahmed	.....	.....
Head of Department	Signature	Date

Abstract

The Internet of Things (IoT) is a new paradigm that enables the convergence of smart objects and the internet. It is an intelligent network that connects all things to the Internet for the purpose of exchanging information and communicating through the information sensing devices in accordance with agreed protocols. Aside the various benefits IoT provides, it also presents challenges related to security and privacy. The direct connection of IoT devices to the internet makes them susceptible to several security threats.

Some ongoing projects for enhancing IoT security include methods for providing data confidentiality and authentication, access control within the IoT network, privacy and trust among users and things, and the enforcement of security and privacy policies. However, even with these mechanisms, IoT networks are vulnerable to multiple attacks aimed to disrupt the network. For this reason, another line of defense, designed for detecting attackers is needed. Intrusion Detection Systems (IDSs) fulfill this purpose.

Previous resarch works propose IDSs in relation to IPv6 over Low-power Wireless Personal Area Network (6LoWPAN). However, since IoT will be used in many application domains with different technologies (WiFi, BLE, NFC and Z-Wave), development of IDSs only for 6LoWPAN is insufficient to meet the security needs of every IoT system.

This research work focuses on IDSs for IoT devices that use WiFi technology. No previous works address IDSs for IoT devices that use WiFi technology. The research proposes lightweight intrusion detection algorithms that addresses Man-In-The-Middle (MITM) and Rogue Access Points (RAP) attacks. It goes further to propose an orchestration framework for IoT devices which can be used to logically isolate these devices in instances where vulnerabilities are found on them.

KNUST

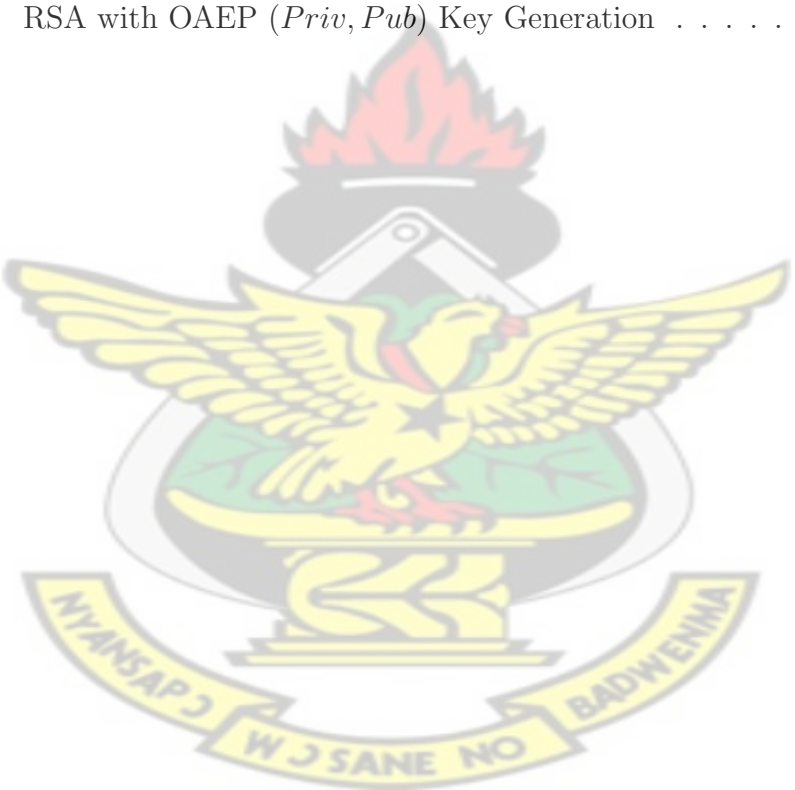
Contents

KNUST

KNUST

List of Algorithms

KNUST



List of Figures

KNUST

KNUST

KNUST

Chapter 1

INTRODUCTION

1.1 Background of Study

The Internet revolution has reinvented business-to-customer (B2C) industries such as media, retail and financial services. This revolution has led to the emergence of Internet of Things (IoT); an ubiquitous global computing network where everyone and everything are connected to the Internet. In an IoT ecosystem, objects in the physical world are embedded with sensors and are connected to the Internet through wireless or wired communication media. These sensors can use various types of local area connections such as RFID, NFC, Wi-Fi, Bluetooth, and Zigbee. Sensors can also have wide area connectivity such as GSM, GPRS, 3G, and LTE. IoT is expected to offer advanced connectivity of devices, systems, and services using a variety of protocols, domains, and applications.

Factors that have led to the rapid development of IoT include: advanced networking capabilities, advancement in connectivity, improvement in cloud computing, availability of low-cost devices and low memory costs [1]. The number of networked devices keeps increasing daily. It is estimated that about 50 billion devices will be connected by the year 2020 (shown in Figure 1.1).

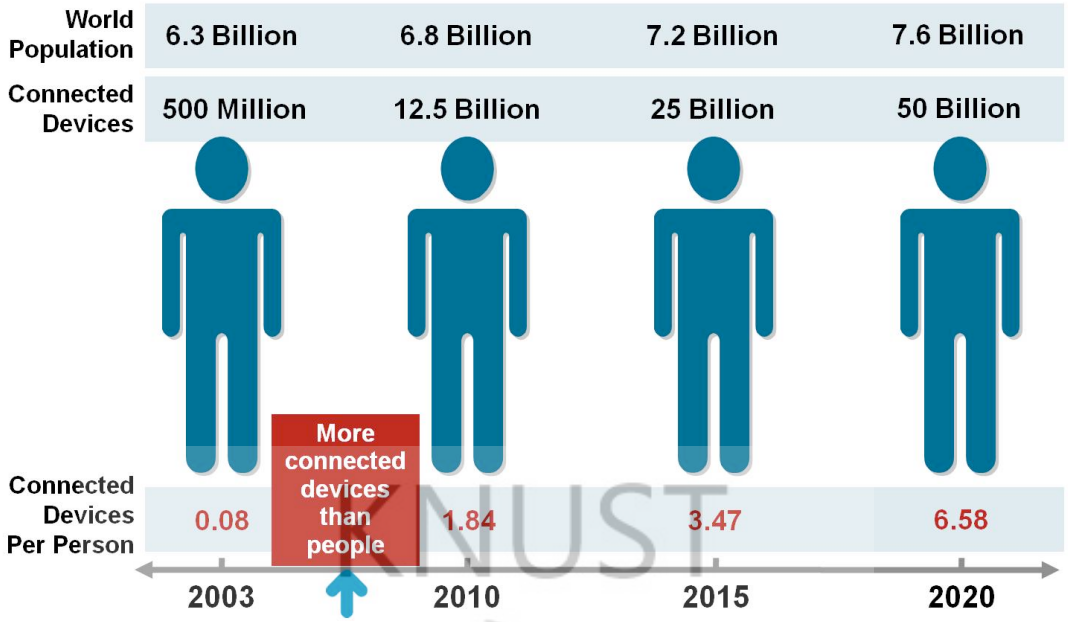


Figure 1.1: The future of connected devices [2].

As of now, IoT is comprised of a collection of purpose-built, disparate networks as shown in Figure 1.2.

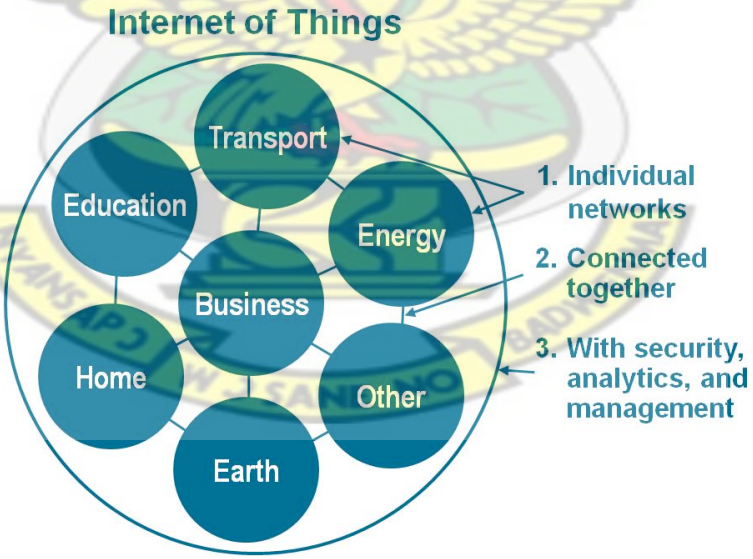


Figure 1.2: IoT Can Be Viewed as a Network of Networks [3].

This new paradigm is perceived as a standout among the most critical actors in the Information and Communication Technology(ICT) industry [4]. It has led

to significant improvements in domains such as public safety, home automation and health care [5].

1.2 Problem Statement

The convergence of IoT devices and the Internet makes them susceptible to being compromised by attackers in situations where vulnerabilities are found on these devices and are not patched. These threats include firmware cloning, firmware replacement, extraction of security information, eavesdropping, Man-in-The-Middle (MITM) attack, routing attack and Denial of Service (DoS) attack [6].

In 2016, the Mirai botnet, also known as *Dyn attack*, took the Internet by storm when it overwhelmed several endpoints with massive Distributed Denial-of-Service (DDoS) attacks [7]. Furthermore, it was reported in 2017 that St. Jude's cardiac devices had vulnerabilities that could allow an attacker to access a device. The battery of these devices could be depleted and also incorrect pacing or shocks can be administered when compromised by an attacker. Devices like pacemakers and defibrillators are used to monitor and control patients' heart functions and prevent heart attacks [8]. The *Owlet WiFi baby heart monitor* was demonstrated to have a poor IoT security. The base station of the device encrypts data sent to and received from the manufacturer's servers, which contact parents' phones if needed. But the Ad-Hoc WiFi network linking the base station to the sensor device is completely unencrypted and doesn't require any form of authentication to access. This allows an attacker to snoop on the wireless data if he's within the connectivity range. The base station creates its own unlocked WiFi network that the sensor (and anyone else) can join. A single unauthenticated command over HTTP can make the Owlet base station leave its current Wi-Fi network and join a malicious one. Hence an attacker can compromise the system and monitor a stranger's baby and prevent alerts from being sent out[9].

Some ongoing projects for enhancing IoT security include methods for providing data confidentiality and authentication, access control within the IoT net-

work, privacy and trust among users and things, and the enforcement of security and privacy policies [10]. However, even with these mechanisms, IoT networks are vulnerable to multiple attacks aimed to disrupt the network. For this reason, another line of defense, designed for detecting attackers is needed. Intrusion Detection Systems (IDSs) fulfill this purpose.

IPv6 over Low-power Wireless Personal Area Network (6LoWPAN) is often cited as a typical IoT network technology. However, since IoT will be used in many application domains with different technologies (WiFi, Bluetooth Low Energy (BLE), Z-Wave, NFC), development of IDSs only for 6LoWPAN is insufficient to meet the security needs of every IoT system.

Most IoT devices use WiFi technology[11]; hence susceptible to conventional wireless attacks. It has been demonstrated that IoT devices are susceptible to MITM attack [12]. This emphasizes the need for measures to be put in place to protect these devices; the development of Intrusion Detection Systems (IDSs).

Conventional techniques used in detecting wireless attacks are not applicable in an IoT scenario due to resource constraints; hence the need for lightweight detection algorithms. Current IoT platforms do not provide a means through which manufacturers can logically isolate devices in instances where vulnerabilities are found on them and need to be patched.

1.3 Research Objectives

1.3.1 General Objective

To develop an Intrusion Detection System (IDS) for WiFi-enabled IoT devices.

1.3.2 Specific Objectives

1. Develop a threat model for WiFi-connected IoT devices.

2. Develop lightweight Intrusion Detection algorithms using the threat model.

3. Test the validity of the proposed algorithm with respect to Man-In-The-Middle (MITM) and Rogue Access Point detection.

4. Develop an orchestration framework for WiFi-enabled IoT devices for logical isolation of vulnerable IoT devices.

1.4 Research Contributions

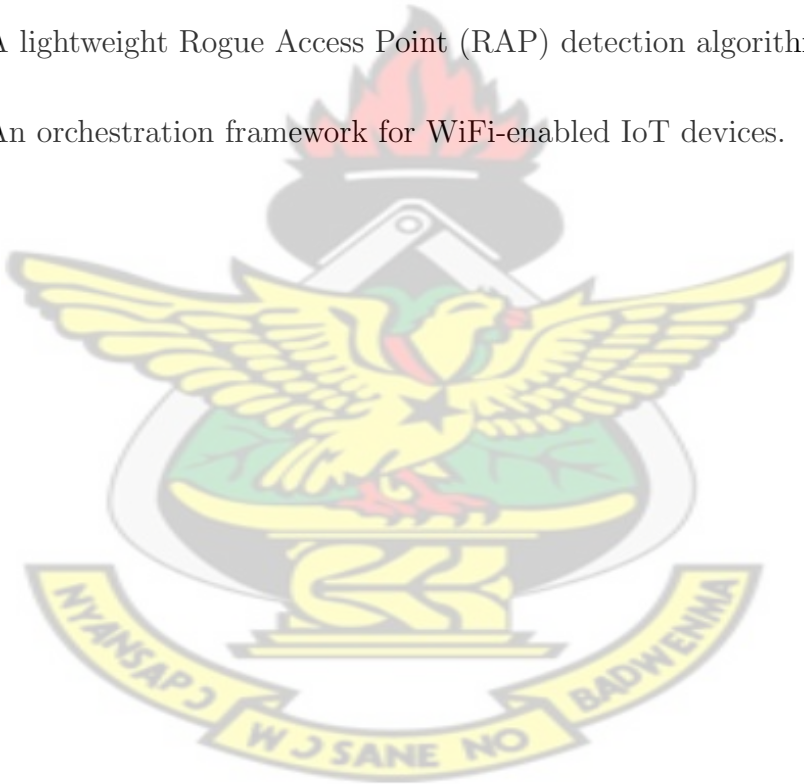
1. Threat model for WiFi-enabled IoT devices.

2. A lightweight MITM detection algorithm.

3. A lightweight Rogue Access Point (RAP) detection algorithm.

4. An orchestration framework for WiFi-enabled IoT devices.

KNUST



Chapter 2

LITERATURE REVIEW

2.1 Introduction

It is required that IoT products have the protection of interaction between IoT entities as a concern in order to improve the security of IoT systems. Auxiliary lines of defense like intrusion detection systems (IDSs) are essential to prevent attackers who may attempt to exploit vulnerabilities in IoT devices. This chapter presents the adopted security model and also reviews works related to MITM, Rogue Access Point detection and existing orchestration frameworks.

2.2 Security Threat Model

The threat model that was used is shown in Figure 2.2. The model presents threats at each layer of the Transmission Control Protocol / Internet Protocol (TCP/IP) stack in reference to WiFi-enabled IoT devices. This model was validated through an experimental analysis of the security and privacy issues of WiFi gateways; described in Section 3.1.

Attack at the physical layer occurs when IoT devices are forced to connect to an illegitimate Access Point (AP) after being disassociated from a legitimate AP. The disassociation of IoT devices from the AP occurs at the data link layer. Spoofing of Media Access Control (MAC) addresses leads to MITM attacks which occurs at the network layer. Teardrop attack occurs at the transport layer. In this kind of attack, an IoT devices is flooded with a large number of Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) packets; which leads to a Denial of Service (DoS). Most IoT devices are also prone to malware attacks due to exploited vulnerabilities; this occurs at the application layer.

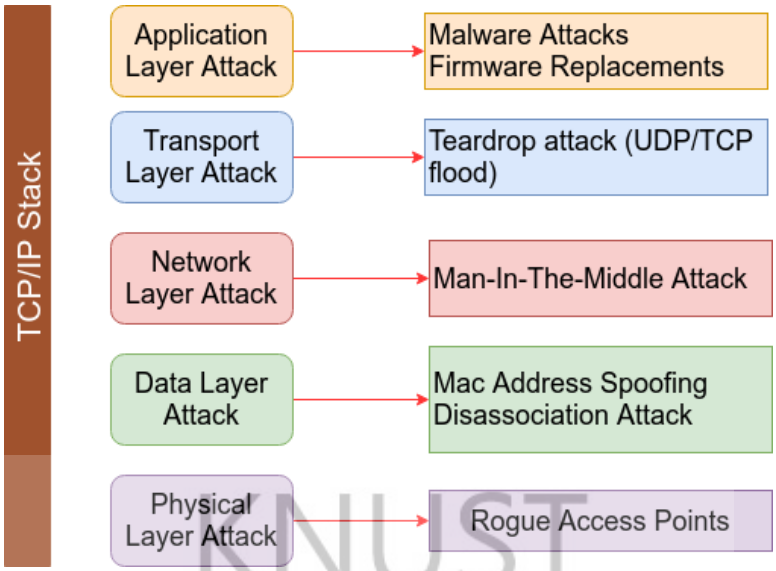


Figure 2.1: Security Threat Model

This research addresses Rogue Access Points (RAP) and MITM attacks which occur at the physical and network layers respectively. The orchestration framework addresses attacks at the application layer through its logical isolation mechanism.

2.3 Man-In-The-Middle (MITM) Attack

MITM is an attack where an attacker impersonates a client node. Considering a communication process between two clients devices A and B, the attacker deceives A by pretending to be B. This enables the attacker to read or modify messages sent from A to B (shown in Figure 2.2). This kind of attack is possible due to a flaw in the Address Resolution Protocol (ARP). The mapping of IP Address to MAC addresses by the data link layer is made possible through ARP[13]. In forwarding a packet from one endpoint to another, the host sending the packet needs to know the recipient's MAC address. Given the IP address of a host, to find its MAC address, the source node broadcasts an ARP request packet to inquire the MAC address of the owner of the IP address. This request is received by all the host devices in the network. The host device that owns that IP address

replies with its MAC address (unicast) [14].

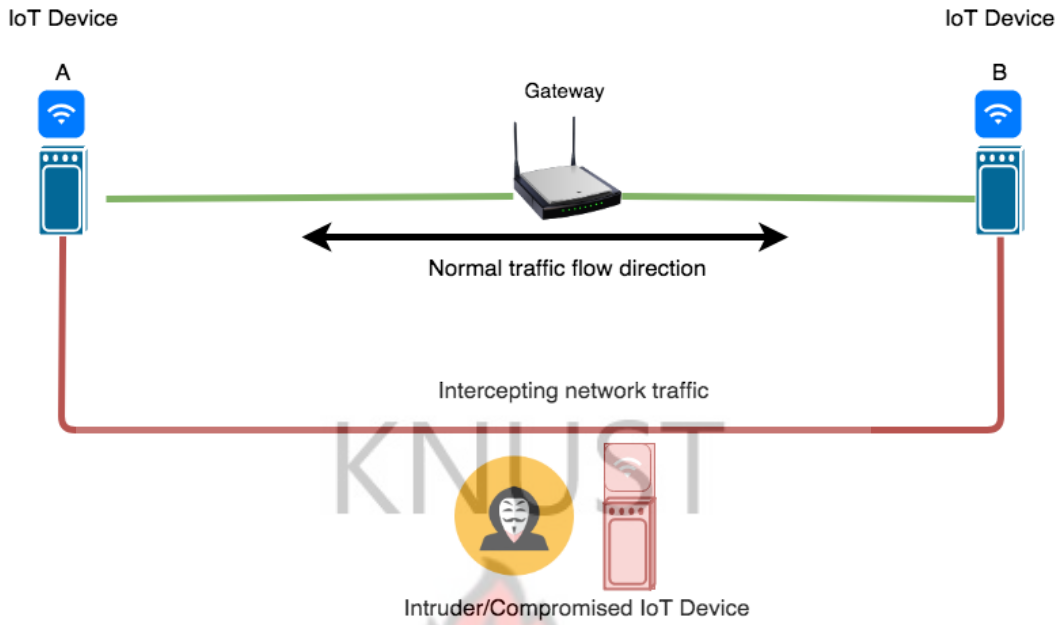


Figure 2.2: MITM Attack

Due to ARP being a stateless protocol and also lacking security in its caching system[15], it accepts ARP replies without considering if an ARP request was sent. This weakness can be exploited by an attacker to initiate MITM attack. A denial of service (DoS) attack can occur if the attacker drops the received packet without forwarding it to the appropriate destination.

Researchers have developed a number of conventional ARP spoofing MITM detection mechanisms. A low-cost embedded IDS capable of detecting and preventing ARP spoofing attacks automatically and efficiently has been proposed [16]. This targets wired Local Area Nework (LAN) environments hence not applicable in a wireless IoT scenario.

A unicast ARP request was proposed instead of the broadcast ARP request by assigning IP addresses via DHCP [17]. The IP/MAC mappings are resolved via DHCP without the need for broadcast. This approach is not applicable for statically assigned IP addresses.

A backward compatible extension to ARP that relies on public-key cryptography to authenticate ARP replies was proposed [18]. All host devices create

public and private key pairs during the initial contact with the network. These keys are then sent to the Authoritative Key Distributor(AKD). This technique leads to performance overhead and it is also not feasible in a wireless network.

A Ticket-based ARP (TARP) that implements security by distributing centrally issued secure IP/MAC ticket via DHCP was proposed [19]. TARP implements security by distributing centrally issued IP/MAC ticket via DHCP. These tickets are sent to the clients when they join the network and are subsequently distributed through existing ARP messages. It leads to performance overhead in generating the public/private key pairs. It is also not suitable for dynamic networks where hosts can join and leave the network anytime.

KNUST

An approach to prevent ARP cache poisoning in wireless LAN by implementing the defense mechanism in the access point (AP) was proposed [20]. The AP constructs the list of correct IP-to-MAC address mapping by monitoring DHCP ACK messages or referring to the DHCP leases file, and blocks all the ARP packets with a false mapping based on the constructed list.

MR-ARP, which is the first voting-based ARP spoofing resistant protocol was proposed [21]. When the MR-ARP endpoint receives an ARP request or reply declaring an (IP, MAC) mapping for a new IP address, it requests neighbour endpoints to vote for the new IP Address. In this mechanism, the voting can be fair only when the voting traffic rates of the responding endpoints are almost the same. This condition can be satisfied in the Ethernet, but may not be valid in the 802.11 network due to the traffic rate adaptation based on the signal-to-noise ratio (SNR).

To overcome the limitation of MR-ARP, EMR-ARP was proposed [22]. This new protocol improves the voting procedure through the implementation of computational puzzles. This mechanism requires too much computational time from devices.

GMR-ARP which is an improvement over EMR-ARP was proposed [23]. It decreased the voting traffic overhead (lower than MR-ARP and EMR-ARP).

Since voting requests are issued in broadcast, this approach can also cause additional overhead.

2.4 Rogue Access Point (RAP) Detection

Rogue access point detection is an import aspect in wireless security. It can be considered as an initial phase of wireless intrusion detection.

A rogue device detection system using techniques such as site survey, Media Access Control (MAC) address list checking, noise checking and wireless traffic analysis has been proposed [24]. The author focused on internal rogue device detection such as wireless devices used by employees in a corporate network. The author used client devices to do periodical scanning to detect rogue access point instead of using dedicated scanning devices. His setup consisted of client devices which communicate with an access point (AP) and the access point communicates to a central server. Captured wireless network traffic is sent by clients through the AP to the central server. Clients transfer the captured data to the AP via hypertext transfer protocol (HTTP) response. The mode of data transmission between the AP and the central server is via eXtensible Markup Language (XML). The central server is responsible for analyzing the received data and upon detecting an intrusion, logs the intrusion to be analyzed by a system administrator. The approach used in this scenario cannot be applied to embedded IoT devices due to resource constraints. Besides, the various client devices used in the periodic scanning are also susceptible to rogue attacks. There is no defense mechanism to protect the client devices used in the periodic scanning of wireless traffic. Furthermore, in situations where client devices are unable to communicate to the central server through the access points, that means rogue access points cannot be detected.

A conventional rogue access point detection system [25] similar to [24] was also proposed. In their approach, wireless access points are manually set to normal operation or sniffing mode. Captured wireless traffic is then stored in a

centralized server and analyzed to detect rogue access points. This approach has limitations similar to that of [24]. Furthermore, it does not incorporate autonomy in its mode of operation, due to the option of manually switching wireless devices modes of operation.

Rogue access point detection based on Received Signal Strength (RSS) was proposed [26]. In this they measured the correlated RSS sequences from nearby APs so as to determine whether the sequences are legitimate or fake. This method works in three phases: The first phase involves the collection of all the RSS from nearby APs. In second phase all these collected RSS are normalized and it estimates the missed RSSs, caused by some external factors and then estimated RSSs are normalizes for generalization of variety of wireless environment. In the last phase, it is determined which RSSs are highly correlated, which is based on some empirical threshold value. The highly correlated RSS sequences are considered as fake signals from a single device.

Mehndi et al. [27], proposed an approach which includes the MAC address, SSID and signal strength of access point in order to decide whether the access point is rogue or not. In detecting authorized access points, the MAC addresses of all visible access points are matched against a list of authorized access points. Tools like Ettercap[28], Wireshark[29] and Snort[30] are further used for filtering in instances where the MAC address is spoofed.

2.5 Orchestration Frameworks

Currently, several IoT frameworks have been developed in order to make it easy to deploy and maintain IoT applications.

Amazon Web Services (AWS) IoT [31] is a cloud-based IoT platform developed by Amazon for easy connection of smart devices to the cloud and also facilitate interaction among these devices. It supports mutual authentication at all points using mechanisms such as X.509 certificates and cognito identities. AWS IoT cloud assigns a private home directory for each legitimate user. All private

data are stored using symmetric key cryptography such as Advanced Encryption Standards (AES)[32].

ARM mbed [33] IoT platform through its ecosystem, provides all requirements needed to develop IoT applications for ARM microcontrollers. It uses Transport Layer Security (TLS)/Datagram Transport Layer Security (DTLS) for communication and authentication purposes.

Azure IoT Suite [34], released by Microsoft, provides a set of services that enables end-users to interact with their IoT devices. It uses TLS for secure communications and SHA-256 for authentication purposes.

KNUST

Google released *Brillo/Weave* [35] platform for the development of IoT applications. *Brillo* is an android operation system for lower power embedded devices where *Weave* is the communication shell for interactions and message-parsing. Link-level security is provided by *Weave* through the use of SSL/TLS protocol.

Calvin [36], an open source IoT platform developed by Ericsson, enables the development and management of distributed applications for IoT devices. Authentication can be done locally or through an external machine and also using a RADIUS server. Secure communication is done using TLS/DTLS protocol.

HomeKit [37] is an IoT framework developed by Apple. It facilitates managing and controlling connected accessories and appliances. Securing communication is achieved using TLS/DTLS with AES-28-GCM and SHA-256.

Kura [38] is an Eclipse IoT project that provides a Java-based framework for IoT gateways that run M2M applications. It consists of security components such as a security service, a certificate service, a Secure Socket Layer (SSL) manager and a cryptography service. All communications are secured using SSL/TLS protocol.

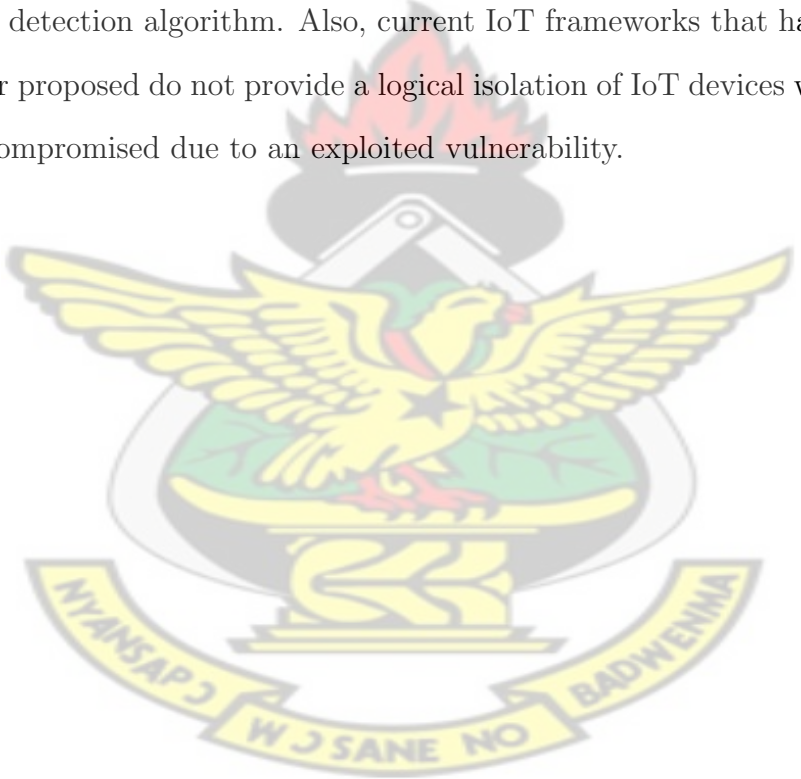
Orchestration systems aimed at providing automated workflow to physical resources (deployment and scheduling) and workload execution management with Quality of Service (QoS) control [39; 1] have been proposed.

Other researchers have also proposed an architecture that allows the orchestration of objects that are part of the Internet of things based on Simple Business Modeling Notation (SBMN) [40] and also orchestration frameworks based on Software-Defined Networking (SDN) [41; 42; 43].

2.6 Summary

From the related works, it can be inferred that the conventional approach used in detecting MITM attacks and Rogue Access Points (RAP) are not applicable to IoT devices due to resource constraints [44]; hence the need for a lightweight MITM detection algorithm. Also, current IoT frameworks that have been developed or proposed do not provide a logical isolation of IoT devices when they have been compromised due to an exploited vulnerability.

KNUST



Chapter 3

METHODOLOGY

3.1 Threat Model

In developing the threat model, three WiFi network Operating Systems were analyzed to determine the extent of their security; OpenWrt, PfSense and Mikrotik Router Operating System(OS). A virtualized network consisting of two client nodes connected to a router as shown in Fig 3.1.

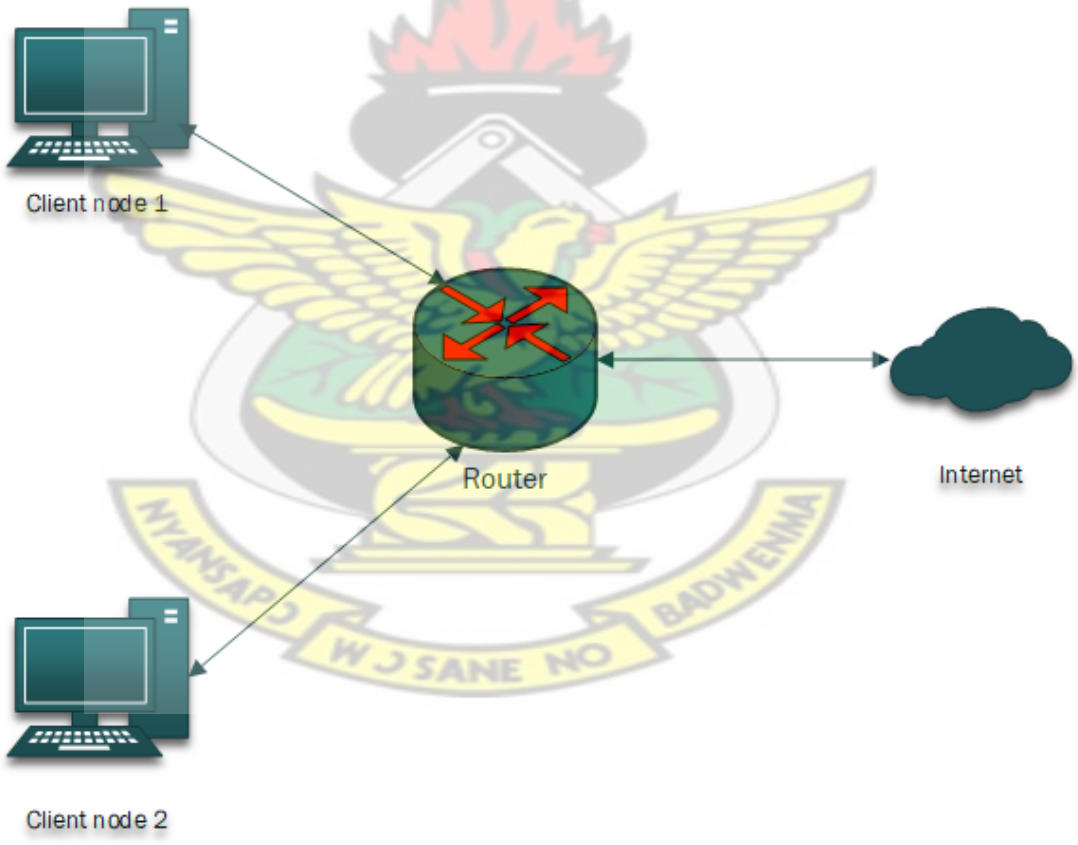


Figure 3.1: Experimental setup

The virtualized router consisted of two network interfaces, the Wide Area Network(WAN) interface and the local network interface. The WAN interface

was configured as a Domain Host Configuration Protocol (DHCP) Client. The router's local network interface was configured as a DHCP Server. Client node 1 acted as a legitimate user and client node 2 acted as a malicious user.

3.2 MITM Attack Detection

The proposed MITM detection and defense algorithm comprises of three sub-level processes coordinated by an InterProcess Controller (IPC) (shown in Figure 3.2).

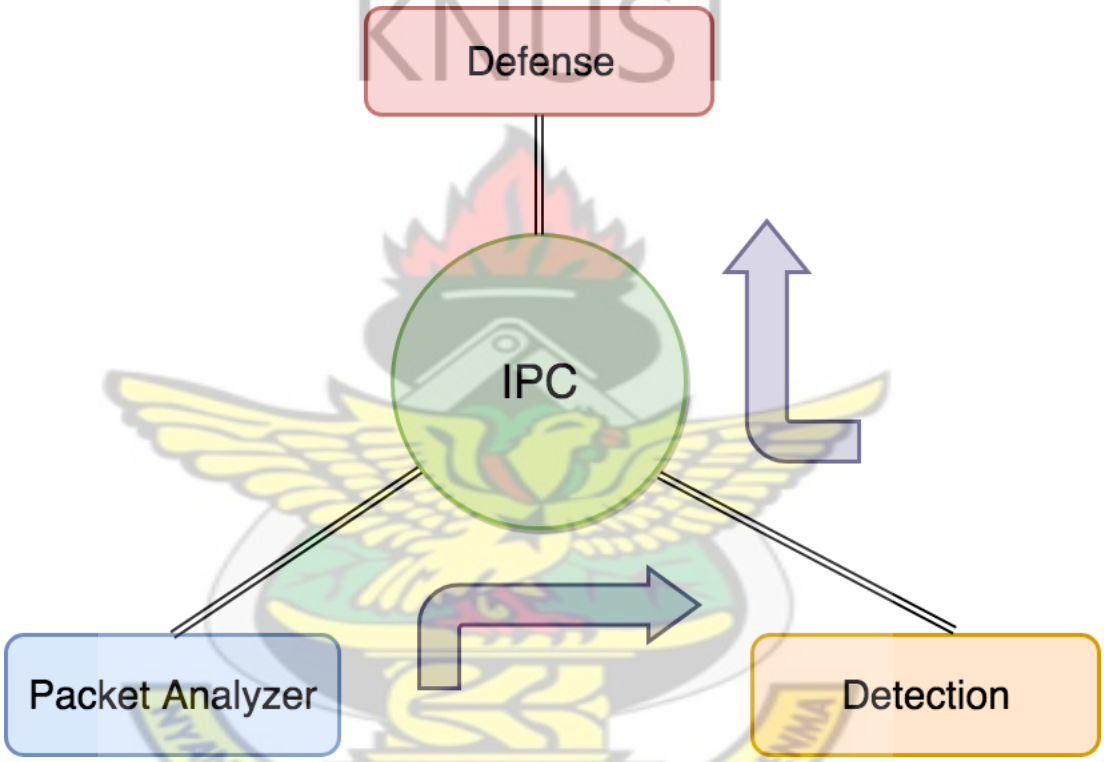


Figure 3.2: MITM Attack Detection and Defense Mechanism

The three sub-level processes consists of the *packet analyzer*, *the detection subprocess* and the *defense subprocess*. The default request and response for HTTP/1.0 is synchronous; hence not appropriate for multiple requests processing. Asynchronous Method Dispatch(AMD) is used in the interprocess communication to enable multiple requests processing.

The algorithm works by first detecting (IP, MAC) mappings of the client nodes connected to the gateway. Legitimate (IP, MAC) mappings are added to

the ARP cache of the gateway which in this case is the AP.

3.2.1 Packet Analyzer

The packet analyzer subprocess is responsible for capturing and decoding wireless traffic. The following packets are captured and analyzed:

- EAPOL/EAP (Extensible Authentication Protocol). The EAP frame is generated as a result of key exchange between client nodes and the AP. The source and destination MAC addresses of the AP and client node are shown in Figure 3.3.
- DHCP. The DHCP frame is generated when the AP issues out an IP address to a client node. The source IP and source MAC address as well as the destination IP and destination MAC address are shown in Figure 3.5.
- IP packet (shown in Figure 3.6).
- ARP. The ARP packet maps client IP addresses to MAC address (shown in Figure 3.4).

For each of the packet frames decoded, the *IP* and *MAC* mappings, together with the time the frame was received, are filtered and passed to the detection subprocess via the interprocess controller.

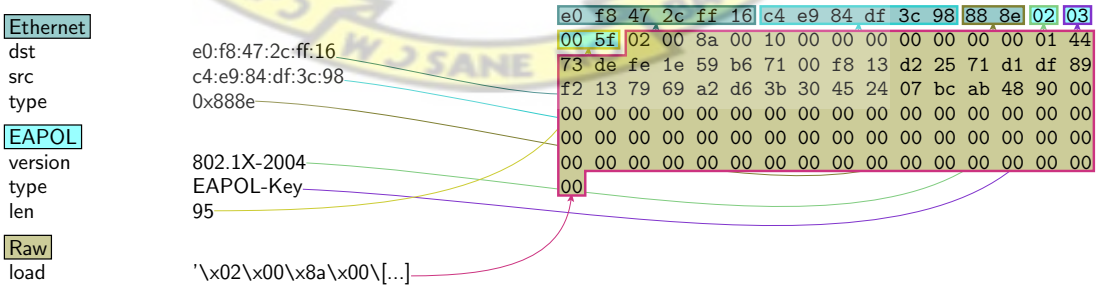


Figure 3.3: EAPOL Decoded Frame

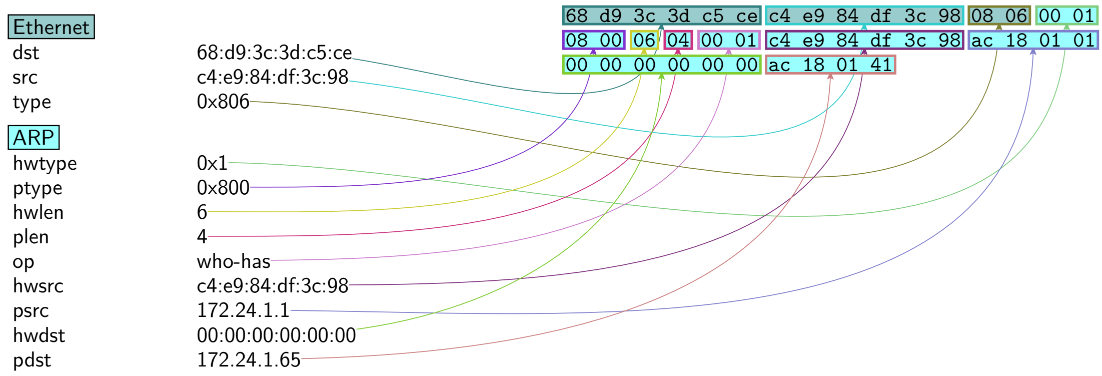


Figure 3.4: ARP Decoded Frame

KNUST

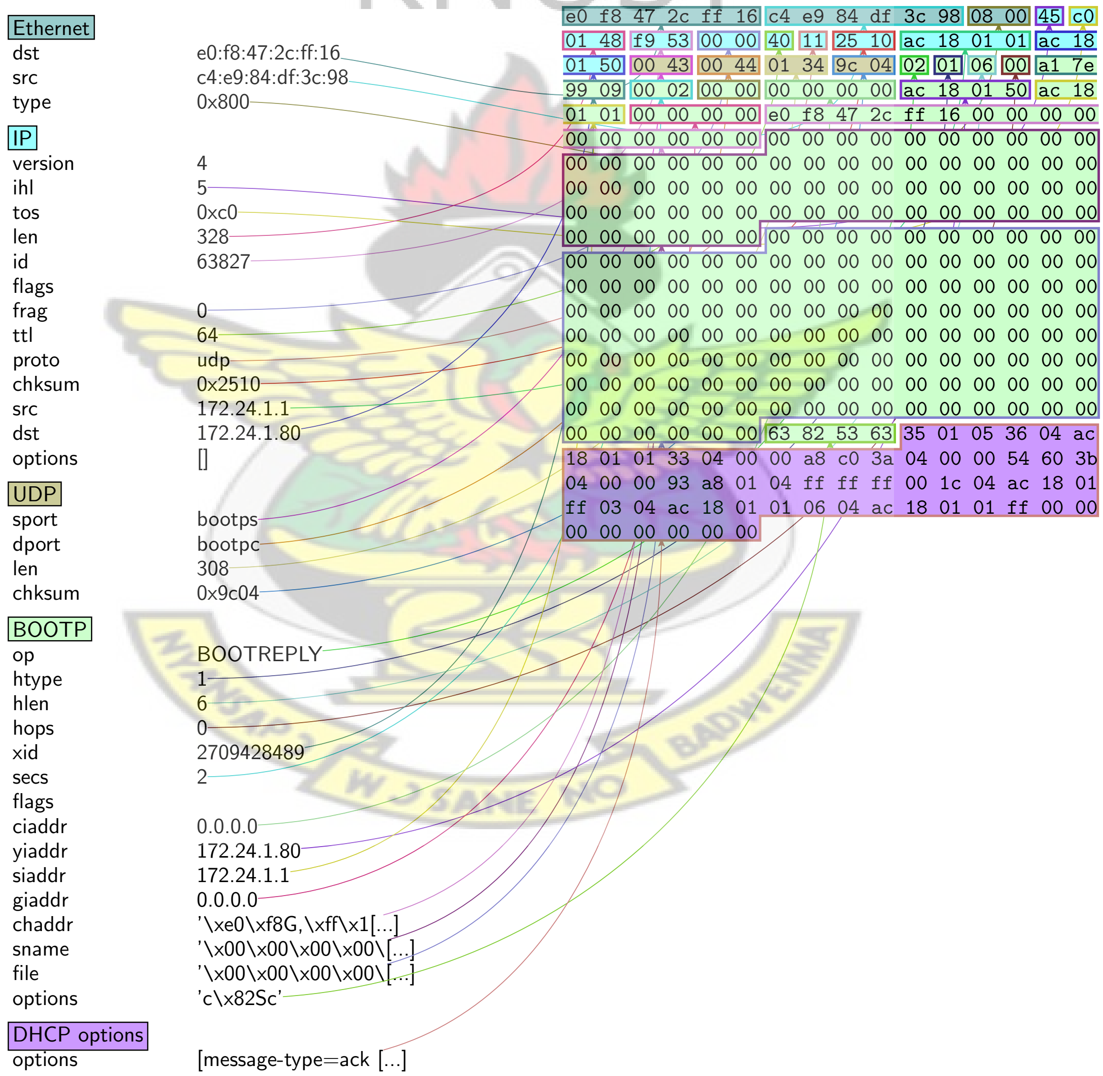


Figure 3.5: DHCP Decoded Frame

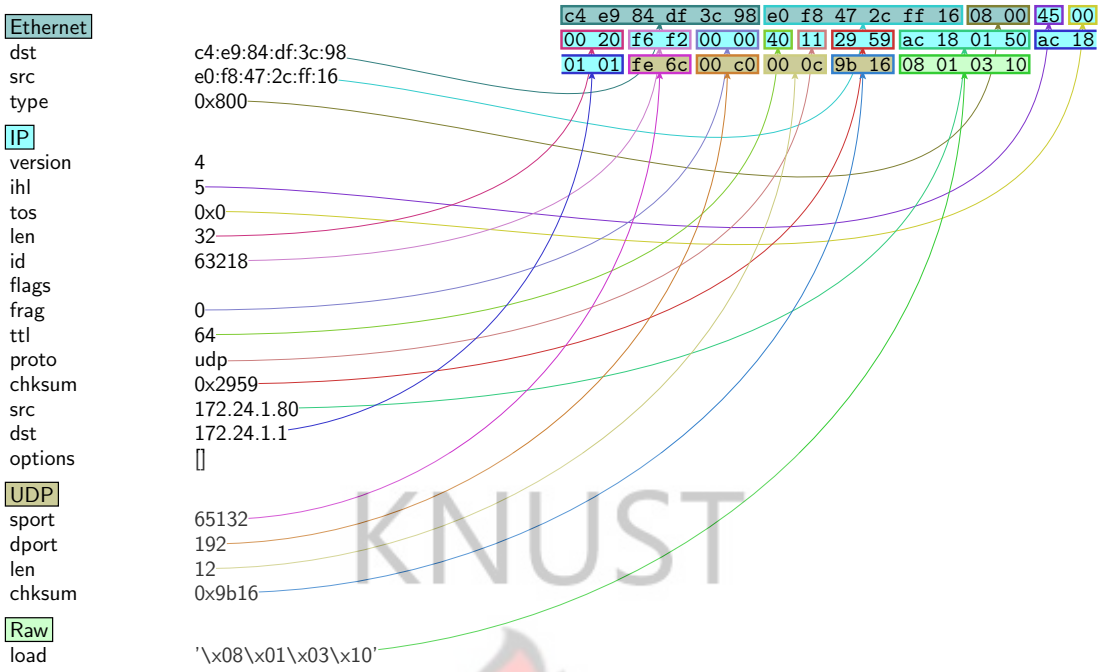


Figure 3.6: IP Decoded Frame

Algorithm 1 Packet Decoding Algorithm

```
while True do
    if sniff(interface) then
        if packet.hasLayer(EAPOL) then
            ← IP, MAC, Time-Seen
        else if packet.hasLayer(DHCP) then
            ← IP, MAC, Time-Seen
        else if packet.hasLayer(IP) then
            ← IP, MAC, Time-Seen
        else if packet.hasLayer(ARP) then
            ← IP, MAC, Time-Seen
        end if
    end if
end while
```

3.2.2 MITM Detection Algorithm

The detection subprocess keeps a virtual ARP cache entries of legitimate IP and MAC mappings and the last time seen. When a new ARP reply is received, the detection subprocess checks the virtual ARP cache to verify if such (IP, MAC)

entry exits. If the entry does not exist, the new (IP, MAC) mapping is added to the virtual ARP cache and also to the ARP cache of the gateway.

If the received (IP, MAC) of an ARP reply contains the same IP and MAC address as one of the entries in the virtual ARP cache, the time seen value of that entry is updated. If the MAC address of an ARP reply is the same as one of the entries in the virtual ARP cache but the IP address varies, the detection subprocess performs an inverse ARP to determine whether the host which previously had the associated MAC address is alive. If the host is alive, then it is flagged as an MITM attack. If the host is not alive, two tests are performed. The first test is to determine the number of hop counts by performing a traceroute to the host IP address. If the hop count is greater than 1, that means the traffic is being intercept by an unauthorized client. This is flagged as an MITM attack. If the hop count is 0, then it is possible the host has been denied of service. A second test is performed to validate whether this is MITM attack. The time difference between the last seen MAC address entry in the virtual ARP cache and the incoming ARP reply's time is computed. The incoming ARP reply is flagged as MITM attack if the resulting time difference is less than the ARP cache entry's time-to-live (TTL).

3.2.3 MITM Defense Algorithm

When a particular ARP reply is flagged as MITM attack, the defense subprocess deletes the IP address of the host performing the spoofing attack from the ARP entry and blocks all traffic originating from that host.

The algorithm was implemented on a Raspberry Pi [45] acting as a gateway for seven IoT devices (NodeMcu[46]) with one of client nodes acting as a malicious client (shown in Figure 3.7).

Algorithm 2 Detection Algorithm

```
1: if arpData then
2:     exists, res = arp.findMac(arpData.mac)
3:     if exists then
4:         if arpData.ip != res.ip then
5:             if InvARP(res.ip) then
6:                 ← Host alive, MITM detected
7:             else
8:                 if hopCount(res.ip) != 1 then
9:                     tDiff = (arpData.t - res.t)
10:                    if  tDiff < ARPTTL then
11:                        ← MITM leading to DoS
12:                    end if
13:                end if
14:            end if
15:        else
16:            ← Update Last Seen
17:        end if
18:    else
19:        ← New ARP Entry
20:    end if
21: end if
```

Algorithm 3 Defense Algorithm

```
1: if mitmData then
2:     deleteARPEntry(mitmData.IP)
3:     dropPackets(mitmData.IP)
4: end if
```

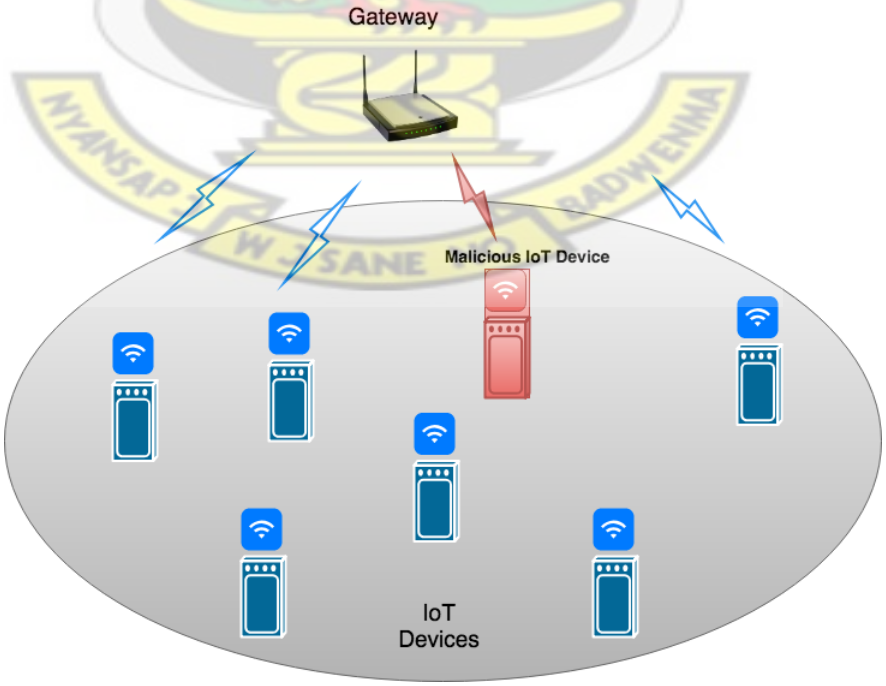


Figure 3.7: Architecture for evaluating MITM Attack

KNUST

3.3 Rogue Acces Point (RAP) Detection

The rogue AP detection algorithm works by profiling the legitimate access point using the following characteristics:

- Basic Service Set Identifier (BSSID)
- Channel
- Frequency
- Protocol
- Cipher
- Supported bit rate(s)

KNUST

The entropy of the legitimate access point is computed using equation 3.1.

$$H(S) = -\sum_{n=0}^{N} p_i \log_2 p_i \tag{3.1}$$

Each of the characteristics is assigned a probability by applying the principle of indifference(Bayesian non-informative prior).

$$P_i = \frac{1}{N} \quad , N > 1 \tag{3.2}$$

N is the number of characteristics under consideration. The entropy is used to quantify the authenticity of an AP based on the characteristics. The algorithm identifies a particular AP as rogue or not by computing its entropy using the same characteristics and comparing the computed entropy with that of the legitimate AP.

3.3.1 Lightweight Rogue Access Point(RAP) Detection Algorithm

The RAP detection algorithm was implemented in two modes. The first mode uses a parser[47] that uses a Linux operating system utility called *iwlist* [48]. The tool enables scanning using a wireless interface and the parser is able to filter the desired characteristics based on the results obtained from the scan. The second mode uses a monitoring wireless interface in capturing the wireless traffic. The captured data is analyzed in realtime to determine whether a particular AP found is rogue or legitimate.

KNUST

Algorithm 4 RAP Detection algorithm using *iwlist* parser

```
1:  while True do
2:      if scan(interface) then
3:          analyze(res)
4:      end if
5:  end while
6:
7:  procedure SCAN(interface)
8:      ← res
9:  end procedure
10:
11: procedure ENTROPY(AP)
12:     entropy = 0
13:     for i ← 0, n do
14:         entropy += −p_i * log_2(p_i)
15:     end for
16:     ← entropy
17: end procedure
18:
19: procedure ANALYZE(res)
20:     if res.ssid == ap.ssid then
21:         if entropy(res) >= entropy(ap) then
22:             ← rogue
23:         else
24:             ← legitimate
25:         end if
26:     end if
27: end procedure
```

In Algorithm 4, a scan is initiated using the *iwlist* utility. The captured

wireless traffic is then analyzed; indicated by the procedure (line 19). If the SSID of the captured traffic matches the SSID of the legitimate AP, the entropy of the detected AP is computed using the procedure on line 11. The decision rule to flag a particular AP as rogue or not is represented as a unit step function of the form

$$\begin{array}{ll} 1 & newEntropy < deviceEntropy \\ 0 & newEntropy >= deviceEntropy \end{array}$$

KNUST

where *deviceEntropy* is the threshold.

In Algorithm 5, the RAP detection works by scanning and detecting beacon frames that bears the same SSID as the legitimate AP. It decodes the beacon frame and matches the entropy of its characteristics to that of the legitimate AP. A sample decoded packet of a legitimate and rogue AP is shown in Figure 3.8 and 3.9 respectively. The **addr1** field represents a broadcast address, the **addr2** and **addr3** are the MAC address of the AP. The **info** field of the first **802.11 Information Element** contains the SSID of the AP.

Algorithm 5 RAP Detection algorithm using Beacon Frame decoding

```
1: while True do
2:     if sniff(mon) then
3:         analyze(res)
4:     end if
5: end while
6:
7: procedure SNIFF(mon)
8:     ← pkt
9: end procedure
10:
11: procedure ANALYZE(res)
12:     pkt = sniff(mon)
13:     if DecodeFrame(pkt) then
14:         if DecodeFrame.ssid == ap.ssid then
15:             if ent(frame) != ent(dev.ssid) then
16:                 ← rogue
17:             else
18:                 ← legitimate
19:             end if
20:         end if
21:     end if
22: end procedure
23:
24: procedure DECODEFRAME(pkt)
25:     if pkt.hasLayer(Dot11Beacon) then
26:         ← decodePacket
27:     else
28:         ← None
29:     end if
30: end procedure
31:
```

KNUST

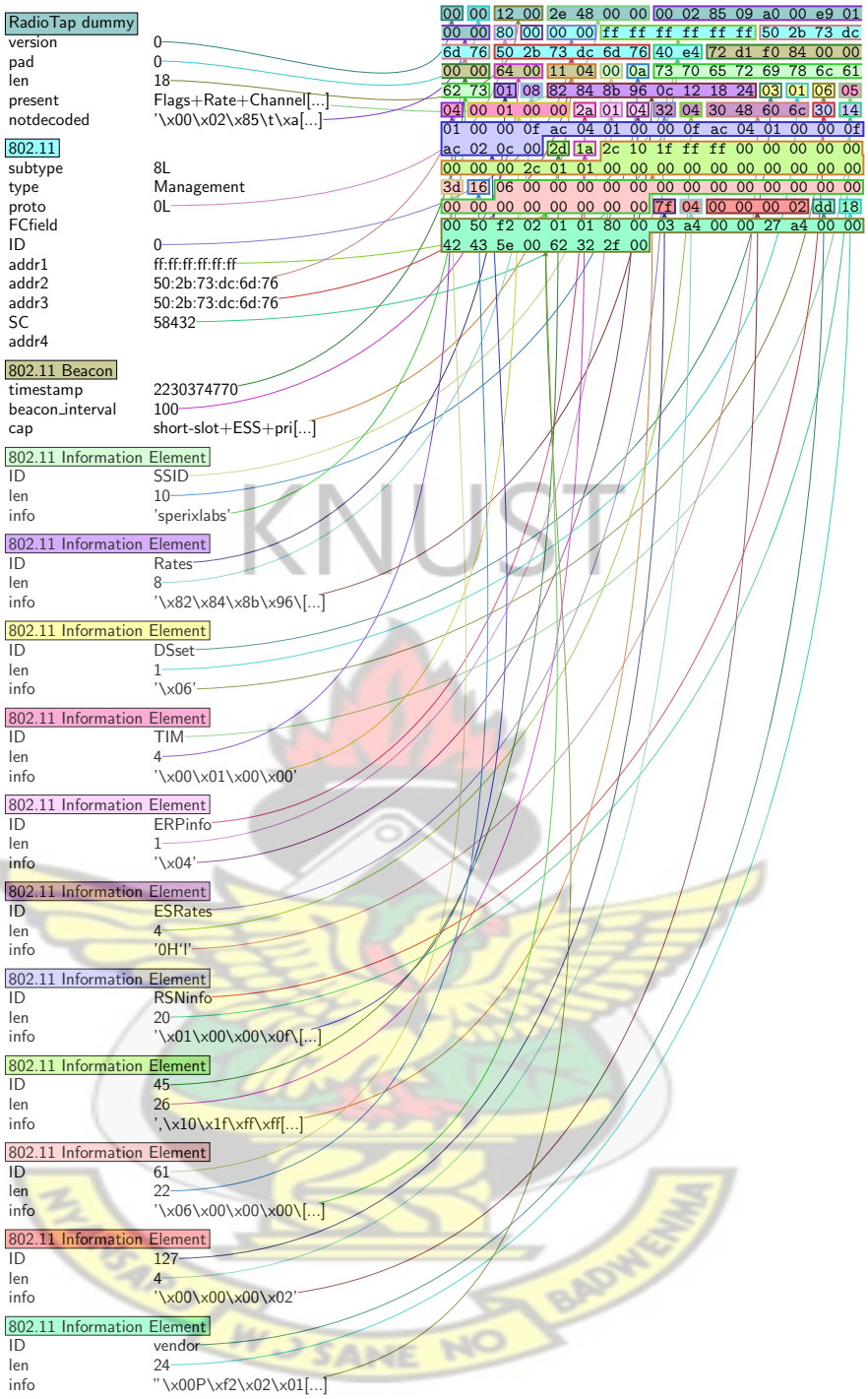


Figure 3.8: A Sample Decoded Beacon Frame from a legitimate AP

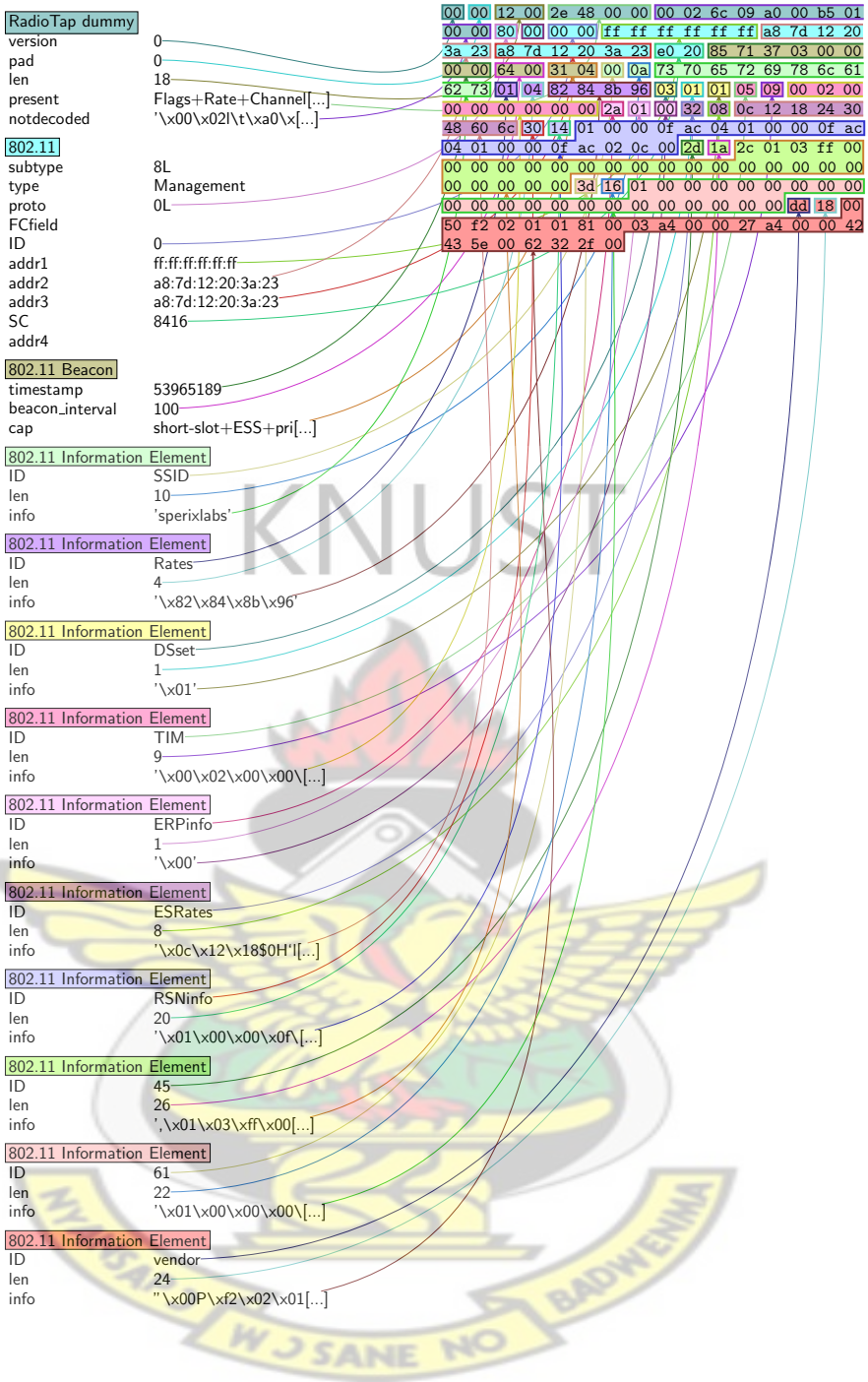
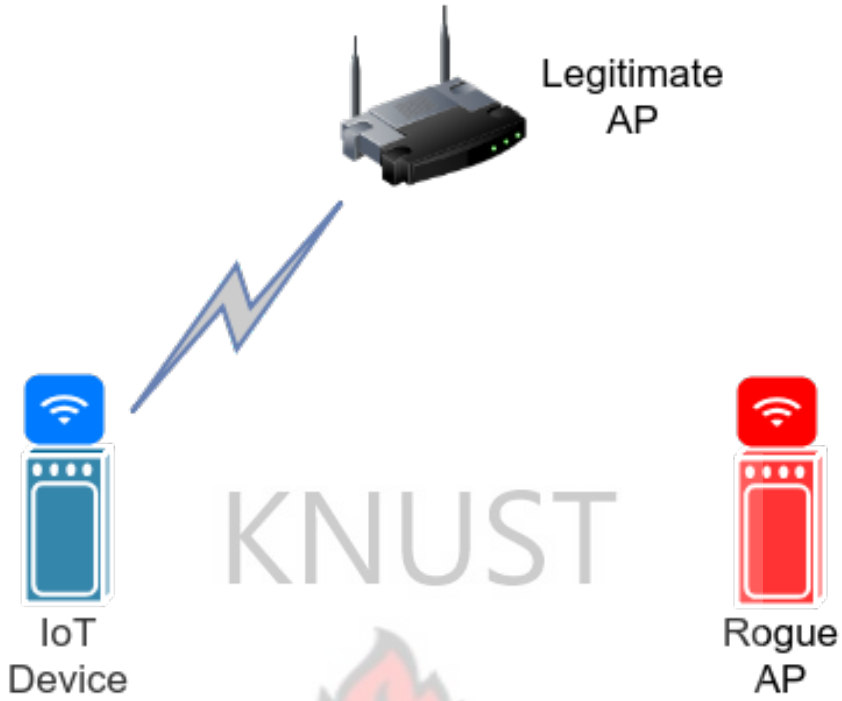


Figure 3.9: A Sample Decoded Beacon Frame from a rogue AP

3.3.2 Experimental Setup

The experimental setup consisted of an IoT device built using a Raspberry Pi 3, a TP-Link AP and a virtualized RAP (shown in Fig. 3.10).



KNUST

Figure 3.10: Experimental Setup for evaluating the RAP Detection Algorithm

The virtualized RAP was setup using a Debian Linux operating system with a Tenda wireless usb adapter. The driver for the wireless usb adapter was recompiled to work with the Linux kernel 4.15.0 [49].

3.3.3 RAP Detection Scenarios

Three scenarios were considered in testing the performance of the RAP detection algorithm. The first scenario implements Algorithm 4 whilst the second and third scenarios implement Algorithm 5 (shown in Fig 3.11).

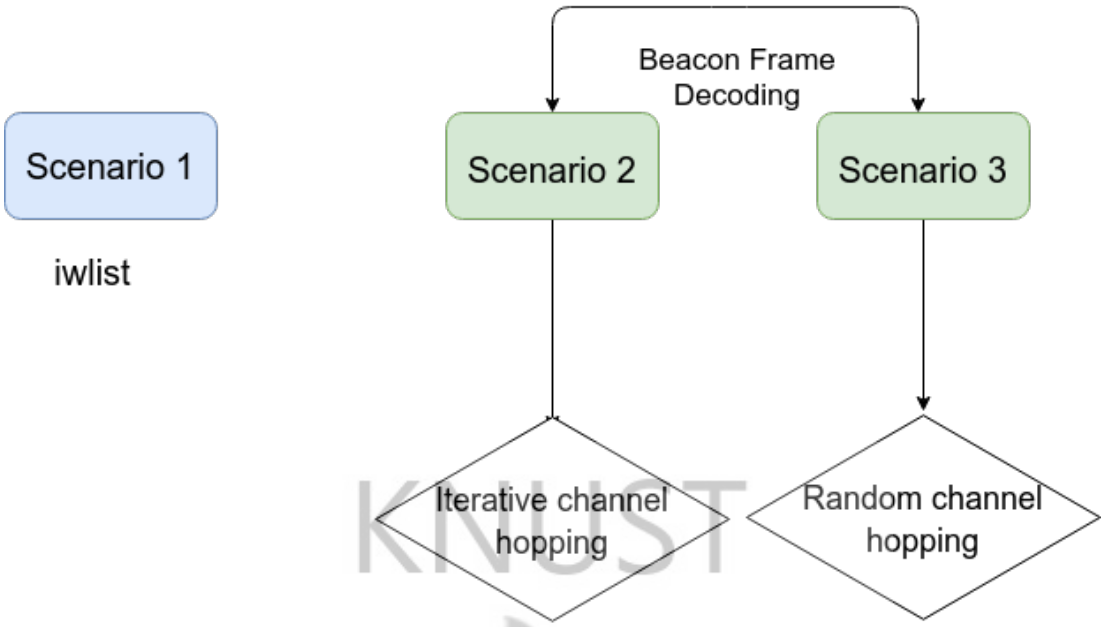


Figure 3.11: Scenarios

In the second and third scenarios, a channel hopping technique (Algorithm 6) is introduced. In scenario 2, a random channel hoping algorithm (Algorithm 6, line 1) was used whereas in scenario 3, an iterative channel hoping algorithm was used (Algorithm 6, line 12).

The detection time and the CPU utilization efficiency was measured while varying the distance between the rogue AP and the legitimate AP. In reference to Figure 3.10, the distance between the legitimate AP and the Rogue AP was varied relative to the position of the IoT device. This was done to verify if the detection algorithm will be able to detect rogues that dynamically change their positions relative to the legitimate AP.

3.4 Orchestration Framework

The orchestration framework employs the use of Public Key Infrastructure in providing logical isolation of IoT devices in instances where these devices are compromised.

Algorithm 6 Channel Hopping Algorithms

```
1: procedure RANDOMHOP(iface)
2:     n = 1
3:     while True do
4:         changeChannel(n, iface)
5:         temp = int(random.random() * 14)
6:         if temp != 0 and temp != n then
7:             n = temp
8:         end if
9:     end while
10: end procedure
11:
12: procedure ITERATIVE(iface)
13:     while True do
14:         for n ∈ (1, 14) do
15:             changeChannel(n, iface)
16:         end for
17:     end while
18: end procedure
```

KNUST

3.4.1 Orchestration Framework Architecture

The entire architecture for the orchestration framework is shown in Figure 3.12.

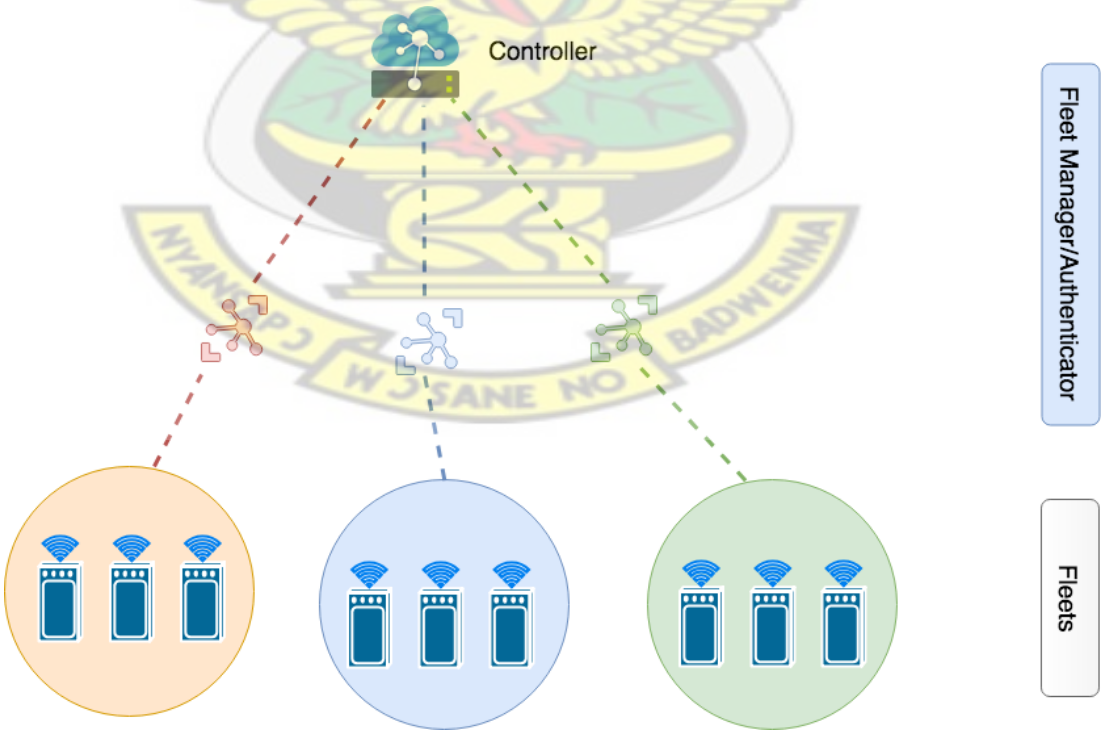


Figure 3.12: Orchestration Framework Architecture

The architecture consists of IoT devices which together form a fleet and are connected to a fleet manager. The various fleet managers are also connected to a main controller. The flow of communication downstream is from the controller to the fleet managers then to the fleet devices. The reverse takes place for upstream communication.

The fleet managers are responsible for authorizing IoT devices belonging to a particular fleet. The controller coordinates and authorizes the various fleet managers. The authorization and management functions of both the fleet managers and the controller can be chained together through Network Function Virtualization (NFV).

KNUST

3.4.2 Flow of Communication

Communication from one endpoint to another is secured using public key cryptography. A pair of asymmetric keys $(Priv, Pub)$ keys are generated at each endpoint. The (Pub) keys are used in encrypting data between endpoints where as $(Priv)$ is used for decrypting received data and also verifying a received data as valid or not (shown in Figure 3.13).

When a fleet manager wants to communicate to an IoT device belonging to its fleet, it encrypts the data using the (Pub) key of the IoT device. The reverse goes for communication between an IoT device and a fleet manager. This is illustrated in Figure 3.14.

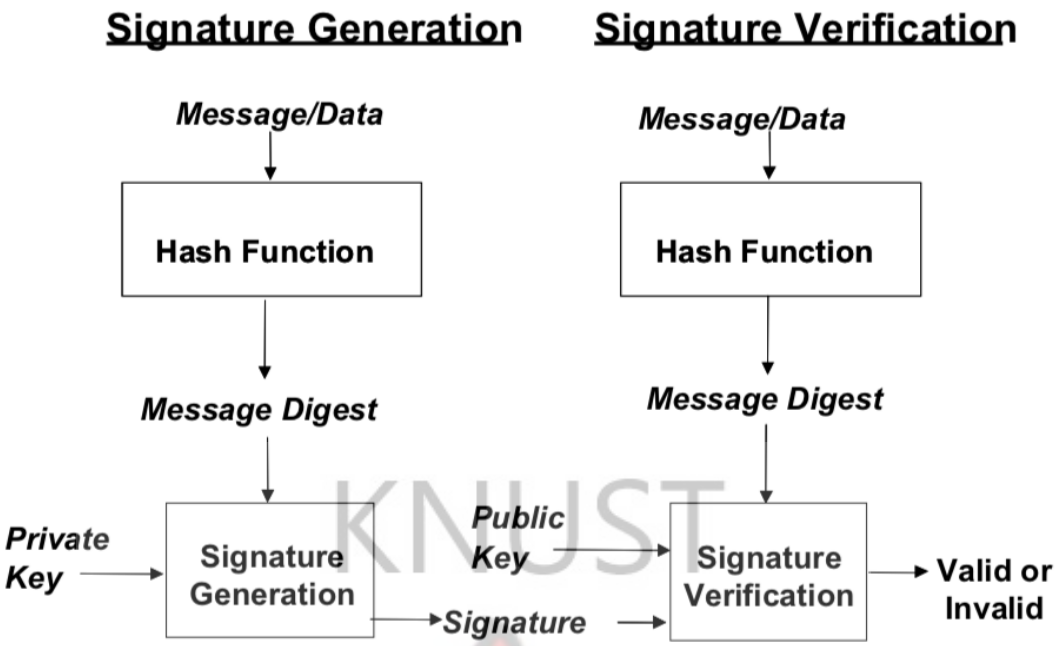


Figure 3.13: Signature Generation and Verification [50]

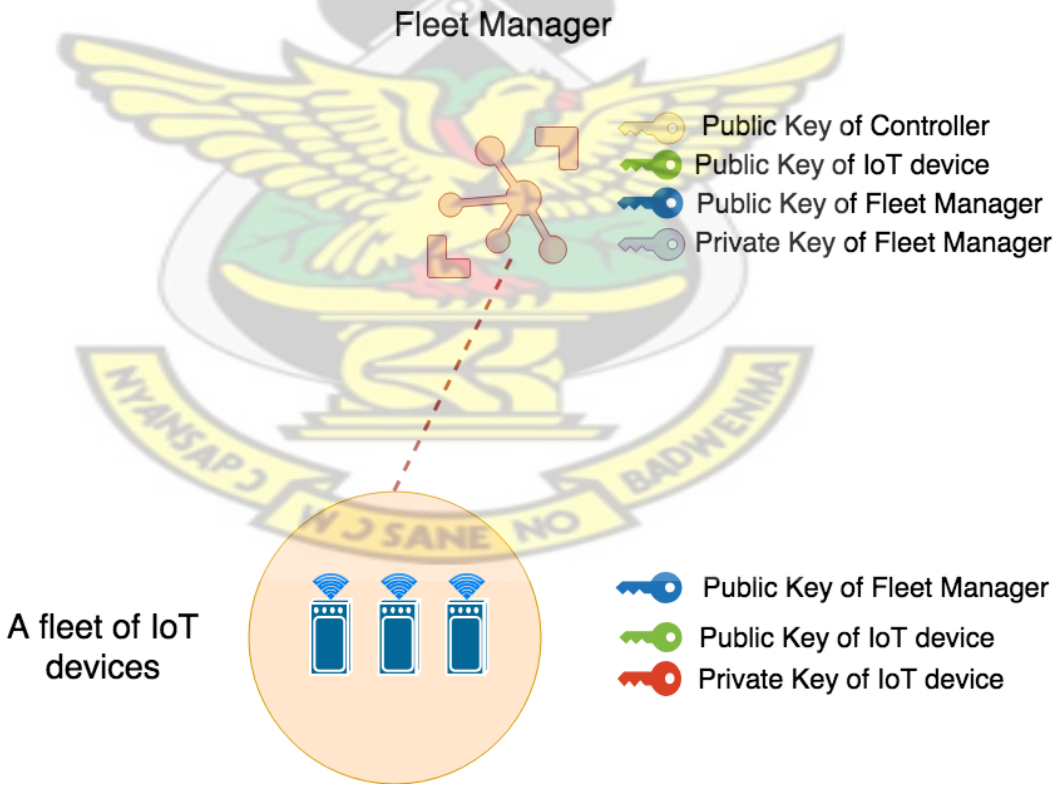


Figure 3.14: Communication between Fleets and Fleet Managers

When a fleet manager wants to communicate to the controller, it encrypts

the data with the controllers (Pub) key. The received data is decrypted by the controller using its ($Priv$) key. The reverse goes for communication between the controller and fleet managers. This is shown in Figure 3.15.

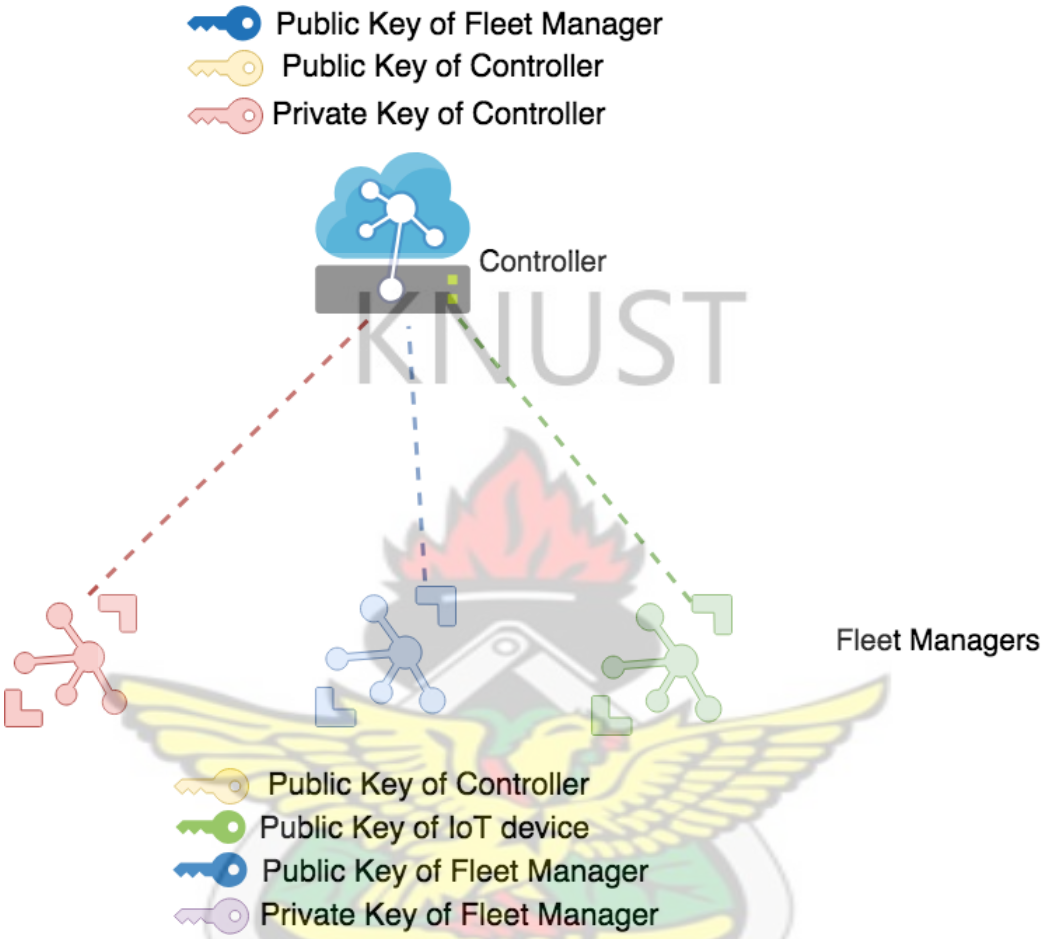


Figure 3.15: Communication between Fleet Managers and Controller

3.4.3 End-to-End Encryption Algorithm for the Orchestration Framework

The orchestration framework implements Rivest-Shamir-Adleman(RSA) cryptography system with Optimal Asymmetric Encryption Padding (OAEP).

RSA cryptosystem generates $(Priv, Pub)$ keys based on the practical difficulty of the factorization problem of the product of two large prime numbers; with a modular exponentiation for all integers m $(0 \leq m \leq n)$, $(m^e)^d \equiv m \, (mod \, n)$, knowing e and n or even m it is extremely difficult to find d.

We employ OAEP because it adds an element of randomness into the RSA algorithm. Also, OAEP prevents partial decryption of ciphertext by ensuring an adversary cannot recover any portion of the plaintext without being able to invert the one-way permutation function. The algorithm for $(Priv, Prub)$ keys generation using RSA and RSA with OAEP is shown in Algorithm 7 and 8 respectively.

Algorithm 7 RSA $(Priv, Pub)$ Key Generation

```
randomSeed = Random.generate()
rsa = RSA.generate(keyLength, randomSeed)
pubKey = rsa.publicKey()
privKey = rsa
```

Algorithm 8 RSA with OAEP $(Priv, Pub)$ Key Generation

```
randomSeed = Random.generate()
rsa = RSA.generate(keyLength, randomSeed)
pubKey = PKCS1_OAEP.new(rsa.publicKey())
privKey = PKCS1_OAEP.new(rsa)
```

3.4.4 Communication Protocol

The framework uses HyperText Transport Protocol (HTTP) with Transport Layer Security (TLS). To overcome the limitation of HTTP being synchronous, we employ the use of websocket for asynchronous requests and responses. This enables a two-way communication process with faster request and response times and also minimal performance overhead.

3.4.5 Message Format

The message format consists of a message token/identifier, a signature verification field and the payload (shown in Figure 3.16).

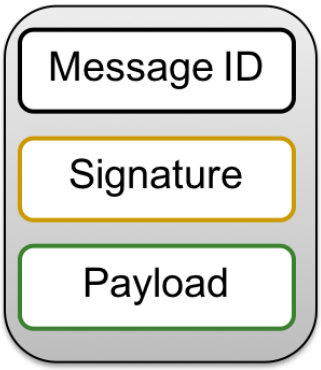


Figure 3.16: Message Format

KNUST

The message token/identifier is generated using the operating system's random number generation algorithm. For operating systems that do not implement the (*os.urandom*) random number generation algorithm, WichmannHill's random number generation algorithm can be implemented. The computational time between the two is quite insignificant (shown in Figure 4.39). WichmannHill's algorithm averages **0.0000751 seconds** whereas the System Random algorithm averages **0.0000597 seconds**. The message token/identifier guards against message replay attacks between endpoints. The signature field is used to validate the endpoints. The payload contains the data to be received by the endpoint; which can be encrypted using the (*Pub*) key of the receiving node.

3.4.6 Evaluation of Proposed Orchestration Framework

The entire orchestration framework was implemented using a RaspberryPi [45] as an IoT device belonging to a particular fleet. The fleet manager and the controller were implemented in the cloud using NFV. The following key performance indicators were used in measuring the efficiency of the orchestration framework:

- The computational time of the public key cryptographic algorithm.
- The performance overhead of the algorithm on endpoints (IoT devices).
- The time it takes to revoke and restore an IoT device's access.

Chapter 4

RESULTS

4.1 Threat Model

In reference to the architecture in 3.1, the following router firmwares were used in the vulnerability assessment: OpenWrt, PfSense and Mikrotik

KNUST

4.1.1 OpenWrt

The OpenWrt Project is a Linux operating system targeting embedded devices. It provides a fully writable filesystem with package management[51]. The version used was **Chaos Chalmer 15.05.1**.

Information Gathering

Device fingerprinting was done using Nmap, which is a free security scanner, port scanner and network exploration tool. Figure 4.1 shows the services running on the OpenWrt firmware router.

	Port	Protocol	State	Service	Version
✔	22	tcp	open	ssh	Dropbear sshd 2015.67 (protocol 2.0)
✔	23	tcp	open	telnet	BusyBox telnetd (OpenWRT, telnet disabled)
✔	53	tcp	open	domain	dnsmasq 2.73
✔	80	tcp	open	http	LuCI Lua http config

Figure 4.1: OpenWrt: Information Gathering

From the information gathered, it was found out that the firmware had services such as Secure Shell(ssh) service, Telnet service, Domain Name System service and an HTTP service running. An initial setup of the firmware required

the use of the telnet service. This allowed the default password to be set. The Telnet service disables automatically after the user password is set.

Authentication Bypass

The administrative page for OpenWrt was not encrypted. It used the standard Hypertext Transport Protocol(HTTP). The user management user interface is shown in Fig 4.2.

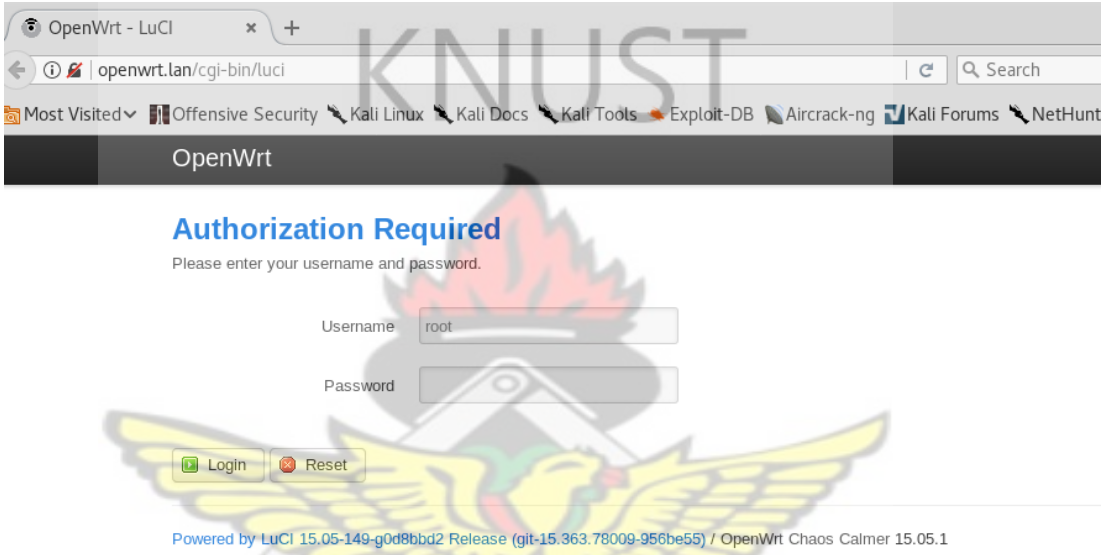


Figure 4.2: OpenWrt: Management Interface

By default the username was **root** which is the superuser in a linux system.

The entire network was sniffed for packet data using Wireshark. It was realized that when a user logs in through the management interface, the login credentials can be sniffed since the HTTP service is unencrypted. This is shown in Fig 4.3.

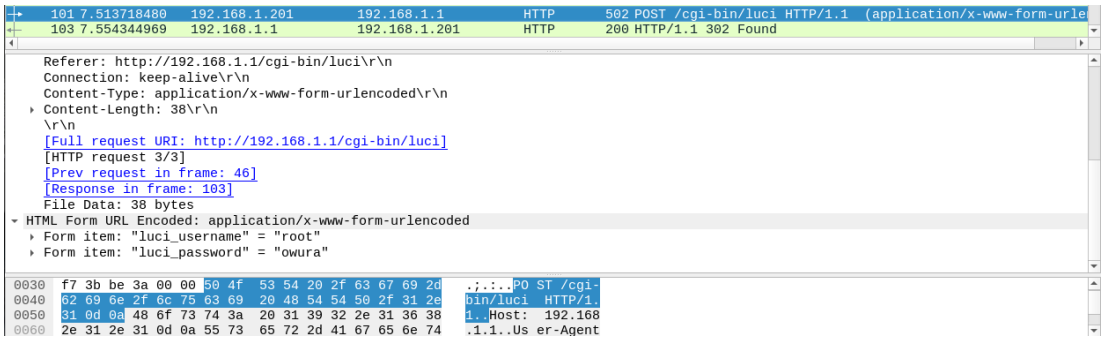


Figure 4.3: OpenWrt: Credentials Capture

When the user is successfully authenticated, the system generates a token together with a cookie which is stored in the user's browser. This was sniffed and captured as shown in Fig 4.4.

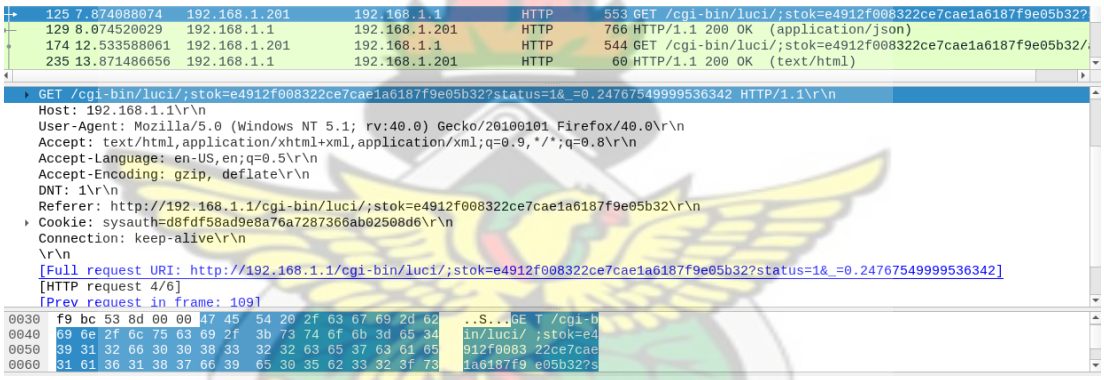


Figure 4.4: OpenWrt: Authentication Token

The authentication was bypassed using the cookie and token generated. This was done by creating a script that sets a cookie with an attribute called **sysauth** and with its value the same as the one captured (shown in Fig 4.5).

```php
<?php
        $cookie_name = "sysauth";
        $cookie_value = "d8fdf58ad9e8a76a7287366ab02508d6";

        setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");
```

Figure 4.5: OpenWrt: Cookie Script

Once the script was executed, the token url was pasted in a browser and

the management interface loaded without any authentication. This is shown in Figure 4.6. The session remains active until the legitimate user logs out.

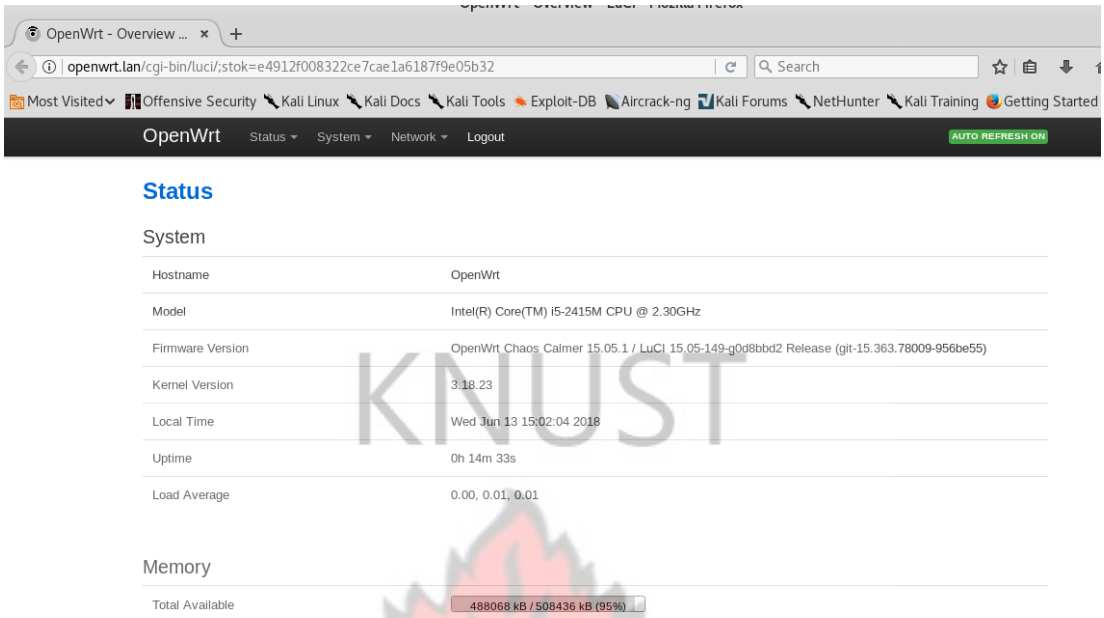


Figure 4.6: OpenWrt: Authentication Bypass

Hosts Discovery

The OpenWrt firmware exposes the identity of client nodes; hostname, Internet Protocol(IP) address and Media Access Control (MAC) address. This can been seen in Fig 4.7 which shows a JavaScript Object Notation(JSON) format of the clients information.

```
HTTP/1.1 200 OK
Connection: Keep-Alive
Transfer-Encoding: chunked
Keep-Alive: timeout=20
Content-Type: application/json
Cache-Control: no-cache
Expires: 0

334
{"swap":{"free":0,"total":0},"conncount":6,"leases":[{"expires":
24002,"macaddr":"08:00:27:c5:0d:1c","ipaddr":"192.168.1.162","hostname":"sperixlabs"},
{"expires":
39639,"macaddr":"08:00:27:6d:a3:13","ipaddr":"192.168.1.201","hostname":"WinXPBE-22020
8"}],"leases6":[{"expires":
24006,"hostname":"sperixlabs","duid":"00043eed2367d6918fa5671bdf8d283083be","ip6addr":
"fde4:b080:a1b2::bf5/128"}],"memory":{"buffered":815104,"total":520638464,"shared":
602112,"free":484823040},"uptime":12661,"wifinets":{},"wan":
{"proto":"dhcp","ipaddr":"10.0.3.15","link":"/cgi-bin/
luci/;stok=e824c840b40b4382cd177609151439fa/admin/network/network/
wan","netmask":"255.255.255.0","gwaddr":"10.0.3.2","expires":-1,"uptime":
307,"ifname":"eth1","dns":["192.168.43.1"]},"localtime":"Wed Jun 13 20:20:51
2018","connmax":16384,"loadavg":[192,960,2976]}
0
```


Figure 4.7: OpenWrt: Hosts Identification

The firmware also exposes network traffic and communication protocol of each client node as shown in Fig 4.8.

```
GET /cgi-bin/luci/;stok=e50fe2799b52e997e933457b07fb7c2b/admin/status/realtime/
connections_status?_=0.13611007731070868 HTTP/1.1
Host: 192.168.1.1
User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:40.0) Gecko/20100101 Firefox/40.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Referer: http://192.168.1.1/cgi-bin/luci/;stok=e50fe2799b52e997e933457b07fb7c2b/admin/
status/realtime/connections/
Cookie: sysauth=f48f6ae8b2919f63ec2b50f42b5e526d
Connection: keep-alive

HTTP/1.1 200 OK
Connection: Keep-Alive
Transfer-Encoding: chunked
Keep-Alive: timeout=20
Content-Type: application/json
Cache-Control: no-cache
Expires: 0

FA8
{ connections:
[{"bytes":"70","src":"127.0.0.1","sport":"60299","layer4":"udp","dst":"127.0.0.1","dpo
rt":"53","layer3":"ipv4","packets":"1"},
{"bytes":"10177772","src":"192.168.1.162","sport":"52042","layer4":"tcp","dst":"192.16
8.1.1","dport":"22","layer3":"ipv4","packets":"109213"},
{"bytes":"76","src":"10.0.3.15","sport":"34705","layer4":"udp","dst":"95.46.198.21","d
port":"123","layer3":"ipv4","packets":"1"},
{"bytes":"70","src":"127.0.0.1","sport":"39471","layer4":"udp","dst":"127.0.0.1","dpor
t":"53","layer3":"ipv4","packets":"1"},
{"bytes":"66","src":"127.0.0.1","sport":"60449","layer4":"udp","dst":"127.0.0.1","dpor
t":"53","layer3":"ipv4","packets":"1"},
```

Figure 4.8: OpenWrt: Hosts Traffic

4.1.2 PfSense

PfSense is an open source firewall/router computer software distribution based on FreeBSD. It is installed on a physical computer or a virtual machine to make a dedicated firewall/router for a network[52]. The version used in the experimental setup was **2.4.3-RELEASE-p1** 4.9.

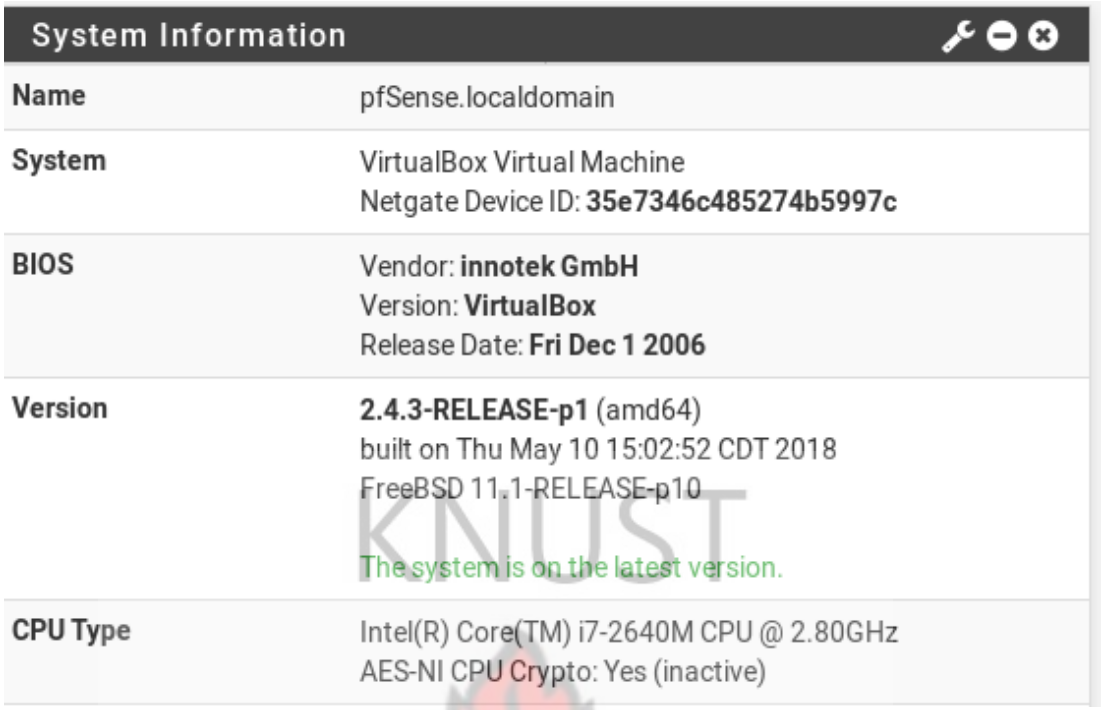


Figure 4.9: pfSense: Version 2.4.3-RELEASE-p1

Information Gathering

Fig 4.10 shows the various services running on the pfSense router.

```
Starting Nmap 7.70 ( https://nmap.org ) at 2018-06-24 04:49 EDT
Nmap scan report for pfSense.localdomain (192.168.1.1)
Host is up (0.00098s latency).
Not shown: 997 filtered ports
PORT    STATE SERVICE  VERSION
53/tcp  open  domain   (generic dns response: NOTIMP)
80/tcp  open  http     nginx
443/tcp open  ssl/http nginx
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-Port53-TCP:V=7.70%I=7%D=6/24%Time=5B2F5B22%P=x86_64-pc-linux-
gnu%r(DNSV
SF:ersionBindReqTCP,
20,"\0\x1e\0\x06\x81\x85\0\x01\0\0\0\0\0\0\x07version\
SF:x04bind\0\0\x10\0\x03")
%r(DNSStatusRequestTCP,E,"\0\x0c\0\0\x90\x04\0\0
SF:\0\0\0\0\0\0");
MAC Address: 08:00:27:BD:88:1F (Oracle VirtualBox virtual NIC)
```

Figure 4.10: pfSense: Information Gathering

The management system was running on port 80 and 443. Also a DNS system was running on port 53.

Authentication Bypass

The default management interface for pfsense(shown in Fig. 4.11) uses HTTP Secure(HTTPS) hence all the transport layer data frames were encrypted.

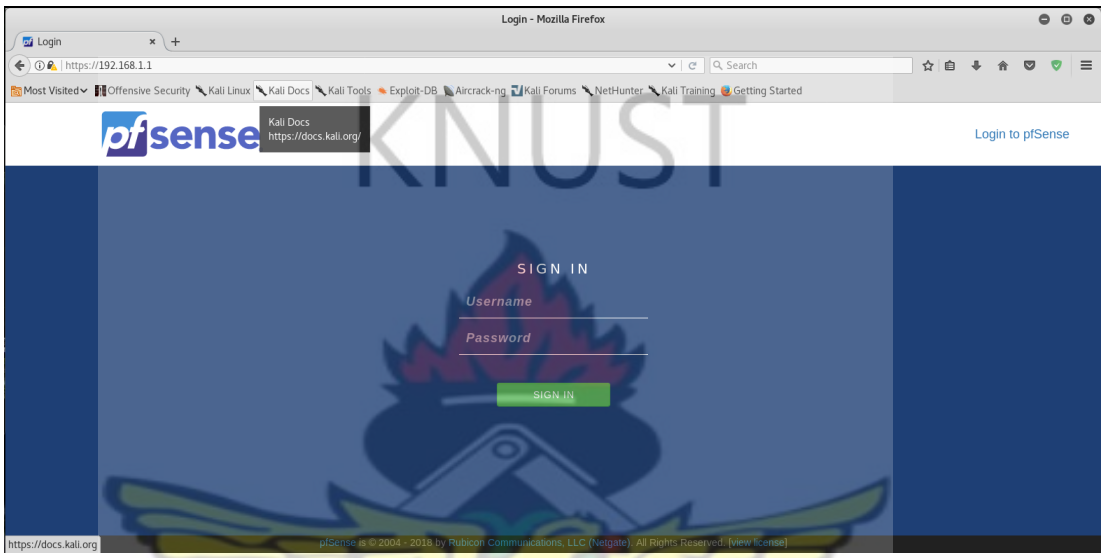


Figure 4.11: pfSense: Management Interface

In order to verify how secure the authentication mechanism was, a man-in-the-middle attack was generated between the client node 1 and the router. Data exchange between the client node 1 and the router were routed through the client node 2. This allowed the client node 2 to strip the HTTPS header from the browser request. Hence all requests and responses received by client node 1 were purely HTTP.

Sniffing network traffic on the local network exposed the credentials used by the network administrator to log in into the management interface. Since the entire traffic was now HTTP, the credentials were harvested as plain text (shown in Fig. 4.12).

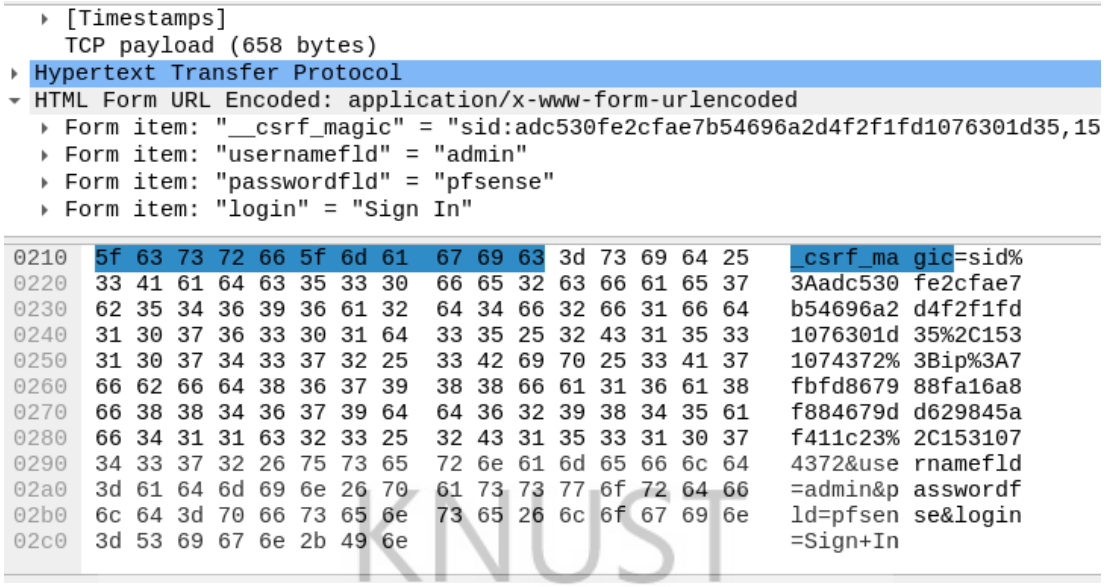


Figure 4.12: pfSense: Harvested Credentials

Also, the firmware stored cookies after the log in credentials had been validated(shown in Fig 4.13). Hijacking this cookie can enable an attacker send requests to the router and retrieve certain details such as system uptime etc (shown in Fig 4.14).

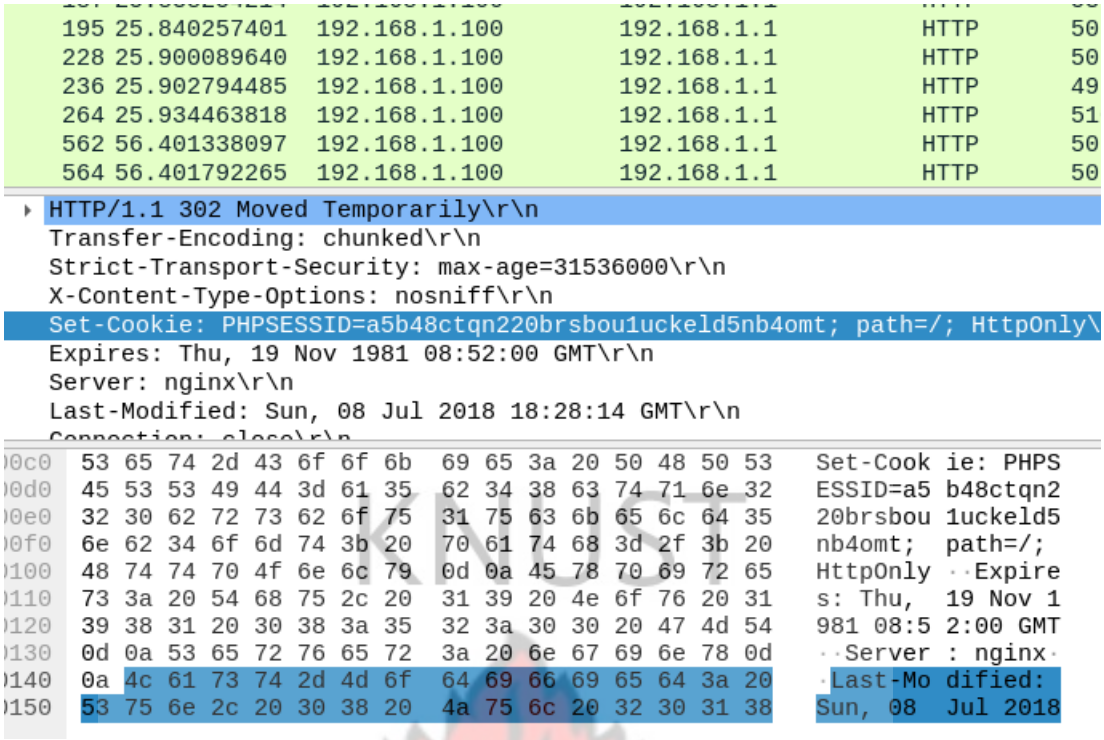


Figure 4.13: pfSense: Session Hijacking

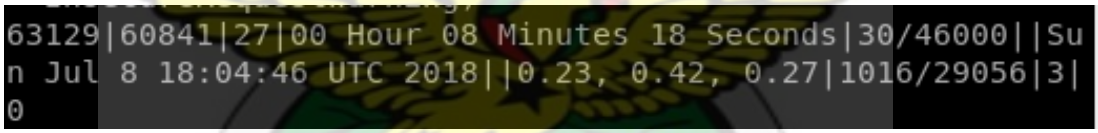


Figure 4.14: pfSense: Information Retrieval through Session Hijacking

Hosts Discovery

With the hijacked session, connected nodes can be identified through a without performing an entire scan of the network. This is shown in Fig 4.15.

Figure 4.15: pfSense: Client Nodes

Review of PfSense Management System

In reviewing the management software of pfSense, it was discovered that the management interface credentials was stored in an eXtensible Markup Language(XML) file. Also the password hashing function used was bcrypt. This is not an efficient way of storing user credentials since it can be bruteforced using rainbow tables (shown in Fig. 4.16).

Figure 4.16: pfSense: Credentials stored in XML file

Also the default credentials is stored on the device in plain text format (shown in Fig. 4.17).

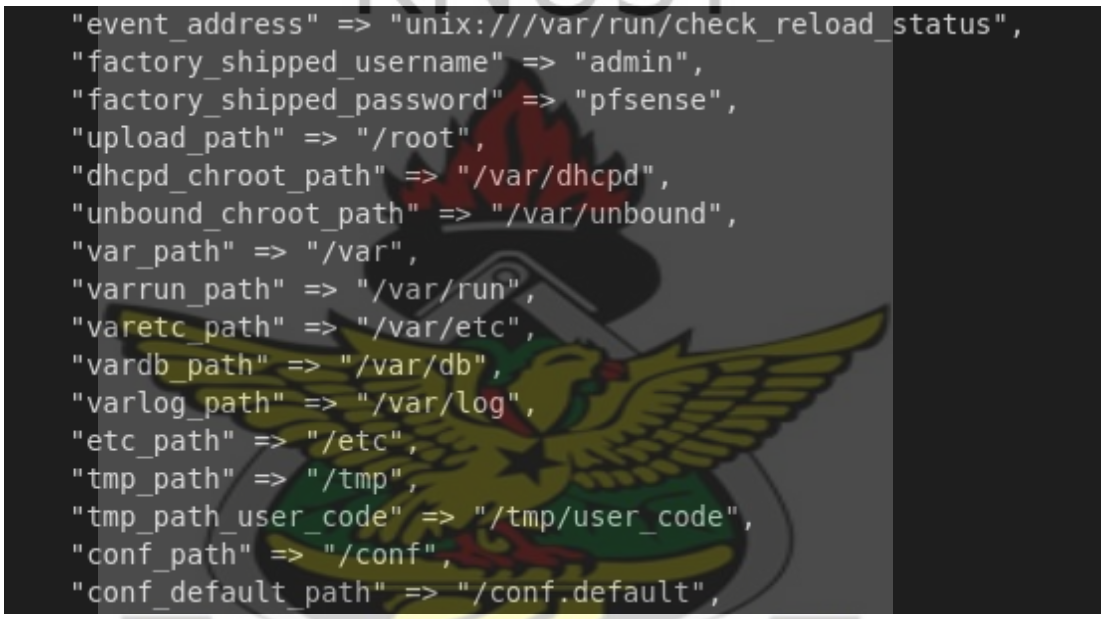
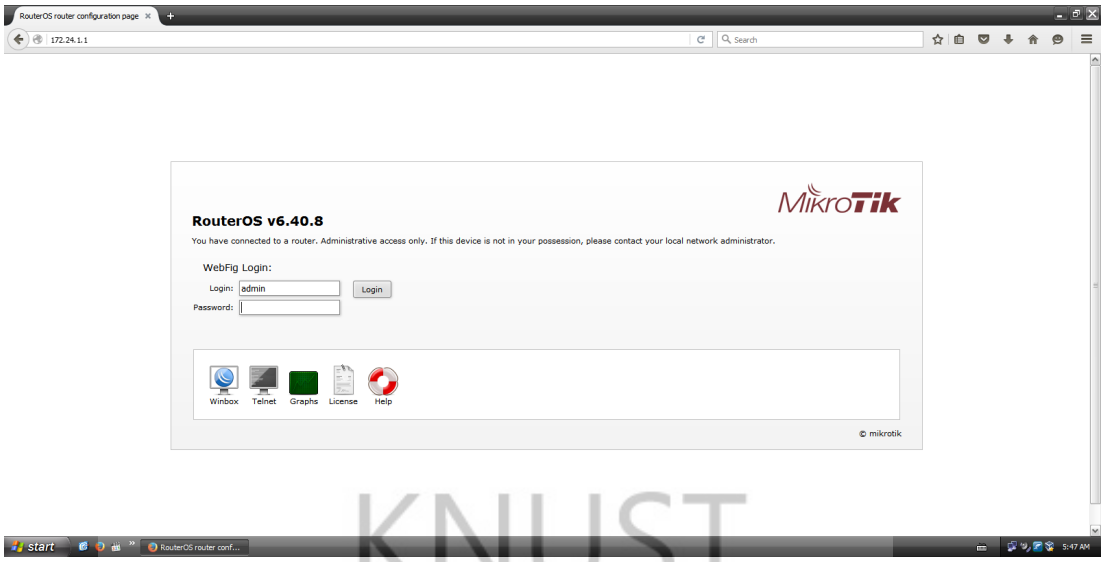


Figure 4.17: pfSense: Default credentials stored in plain text

4.1.3 MikroTik RouterOS

RouterOS is a routing operating system which provides features such as routing, firewall, bandwidth management, wireless access point, backhaul link, hotspot gateway and Virtual Private Network(VPN) server. The version used in the experiment was **v6.40.8** as shown in Fig 4.18.



KNUST

Figure 4.18: MikroTik: v6.40.8

Information Gathering

The following services(shown in Fig 4.19) were found to be running on the RouterOS,

1. File Transfer Protocol(FTP) service.
2. Secure Shell(SSH) service.
3. Telnet service.
4. HTTP service.

Two other services were running on port 2000 and 8291. These were service ports used by the WinBox application client to communicate to the RouterOS.

```
Not shown: 994 closed ports
PORT     STATE SERVICE        VERSION
21/tcp   open  ftp            MikroTik router ftpd 6.40.8
22/tcp   open  ssh            MikroTik RouterOS sshd (protocol 2.0)
23/tcp   open  telnet         Linux telnetd
80/tcp   open  http           MikroTik router config httpd
2000/tcp open  bandwidth-test MikroTik bandwidth-test server
8291/tcp open  unknown
MAC Address: 08:00:27:66:3C:90 (Oracle VirtualBox virtual NIC)
Service Info: OSs: Linux, RouterOS; Device: router; CPE: cpe:/o:mikrotik:routeros, cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 174.66 seconds
```

Figure 4.19: MikroTik: Information Gathering

Authentication Bypass

The default protocol for the management interface is HTTP. The system routes its connection through a Javascript(JS) proxy which tends to obfuscate the data being transmitted (shown in Fig 4.20).

Figure 4.20: MikroTik: JS proxy encrypted data

It makes session hijacking impossible. It was found out that the system sets a cookie in the browser once a user is authenticated (shown in Fig 4.21).

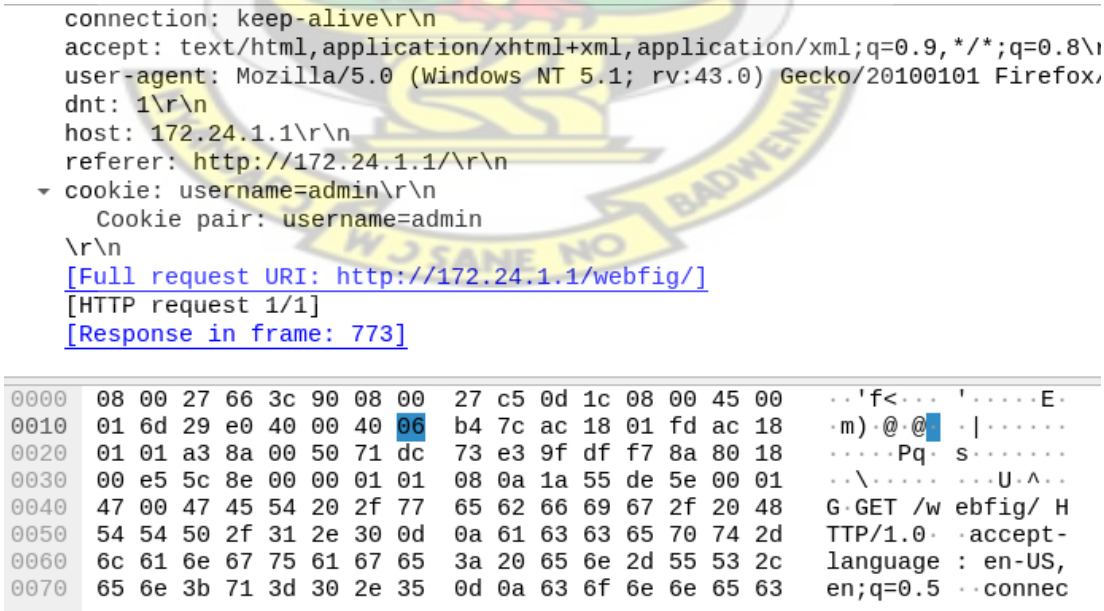


Figure 4.21: MikroTik: Cookie Hijacking

The system could not be bypassed using the hijacked cookie due to the Javascript proxy (JS Proxy) implemented in the web service; the JS Proxy obfuscates session parameters. Due to this, different attack measures were considered.

The system was exploited for vulnerability in the FTP service and the Telnet service. It was possible to sniff the username and password through the FTP service since it was unencrypted (shown in Fig 4.22).

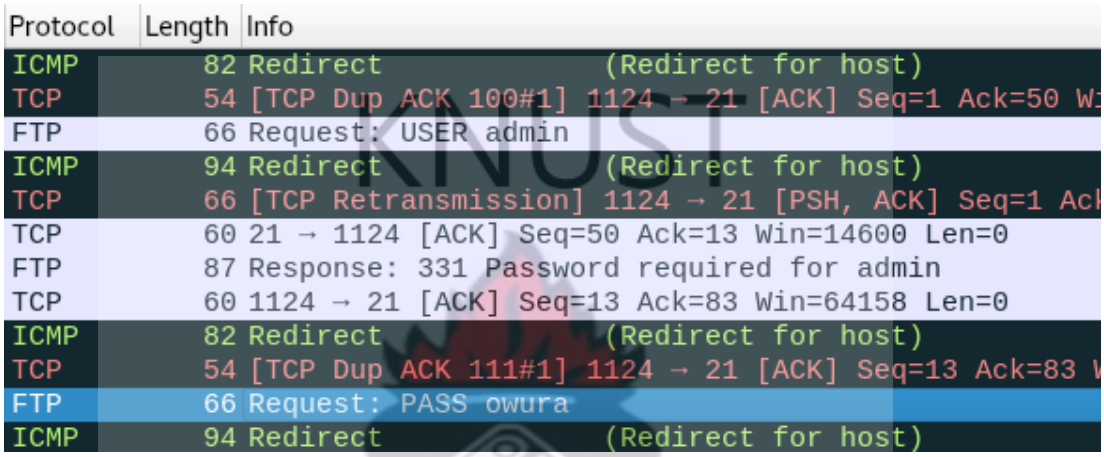


Figure 4.22: MikroTik: FTP Credentials Harvesting

User credentials was sniffed through the Telnet service (shown in Fig).

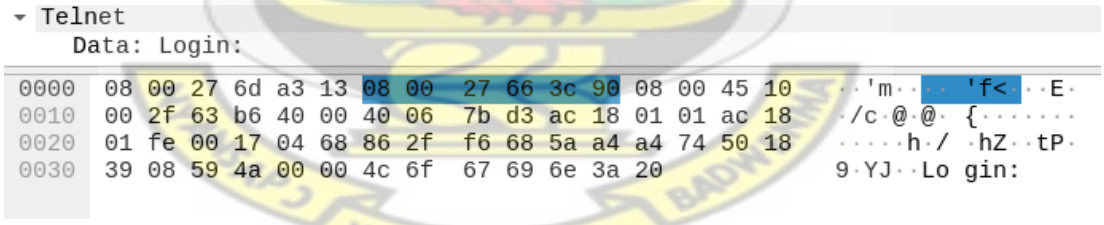


Figure 4.23: MikroTik: Telnet Requesting for Username

```
▾ Telnet
    Data: a

0000  08 00 27 6d a3 13 08 00  27 66 3c 90 08 00 45 10

▾ Telnet
    Data: d

0000  08 00 27 c5 0d 1c 08 00  27 6d a3 13 08 00 45 00

▾ Telnet
    Data: m

0000  08 00 27 c5 0d 1c 08 00  27 6d a3 13 08 00 45 00

▾ Telnet
    Data: i

0000  08 00 27 c5 0d 1c 08 00  27 6d a3 13 08 00 45 00

▾ Telnet
    Data: n

0000  08 00 27 c5 0d 1c 08 00  27 6d a3 13 08 00 45 00
```

KNUST

Figure 4.24: MikroTik: Telnet Credentials Harvesting

```
    TCP payload (12 bytes)
▾ Telnet
    Data: \r\n
    Data: Password:

0000  08 00 27 6d a3 13 08 00  27 66 3c 90 08 00 45 10   ··'m····  'f<···E·
0010  00 34 63 be 40 00 40 06  7b c6 ac 18 01 01 ac 18   ·4c·@·@·  {·······
0020  01 fe 00 17 04 68 86 2f  f6 74 5a a4 a4 7e 50 18   ·····h·/  ·tZ··~P·
0030  39 08 a6 6f 00 00 0d 0a  50 61 73 73 77 6f 72 64   9··o····  Password
0040  3a 20                                             :
```

Figure 4.25: MikroTik: Telnet Requesting for Password

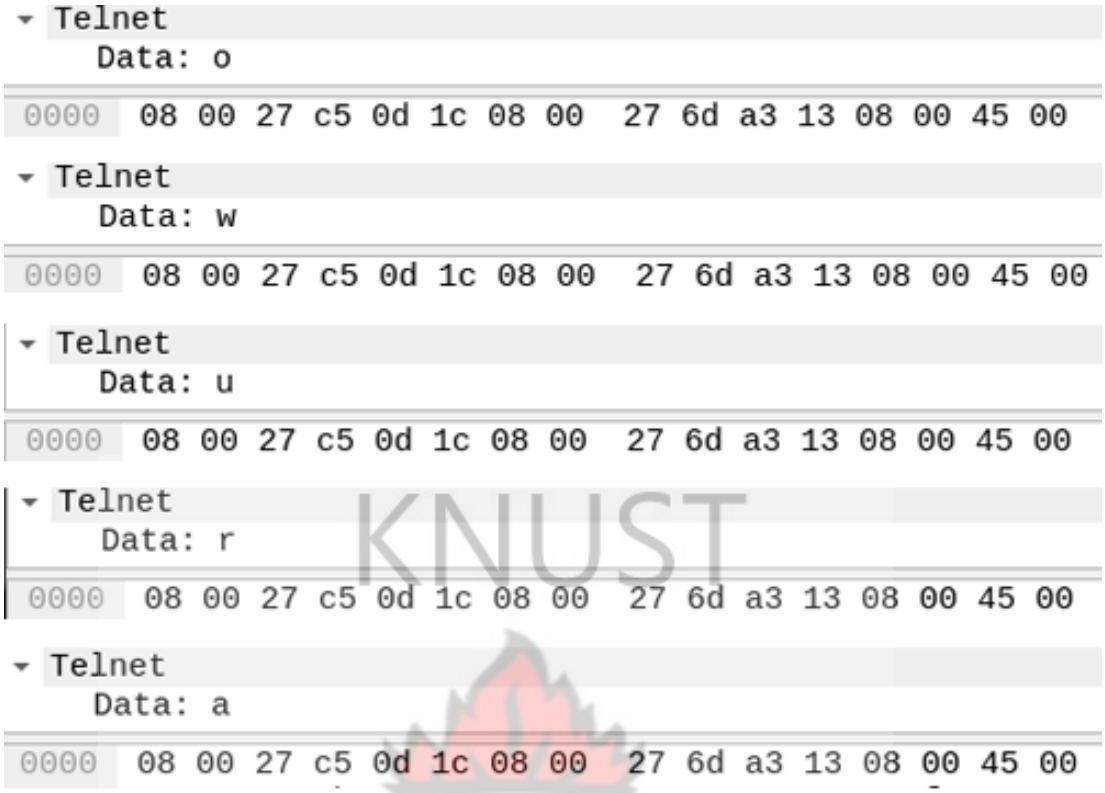


Figure 4.26: MikroTik: Telnet Credentials Harvesting

4.2 MITM

In reference to Figure 3.7 and the methodology described in Section 3.2, three key performance indicators were used in determining the efficiency of the proposed algorithm: CPU utilization efficiency, detection time and network latency (using the round trip time (RTT)). Outliers of the data acquired are as a result of the threading effect on the CPU.

Figure 4.27 shows the performance overhead when the algorithm was implemented; averages **0.9545%**. It outperforms that of [20] which was **1.65%**.

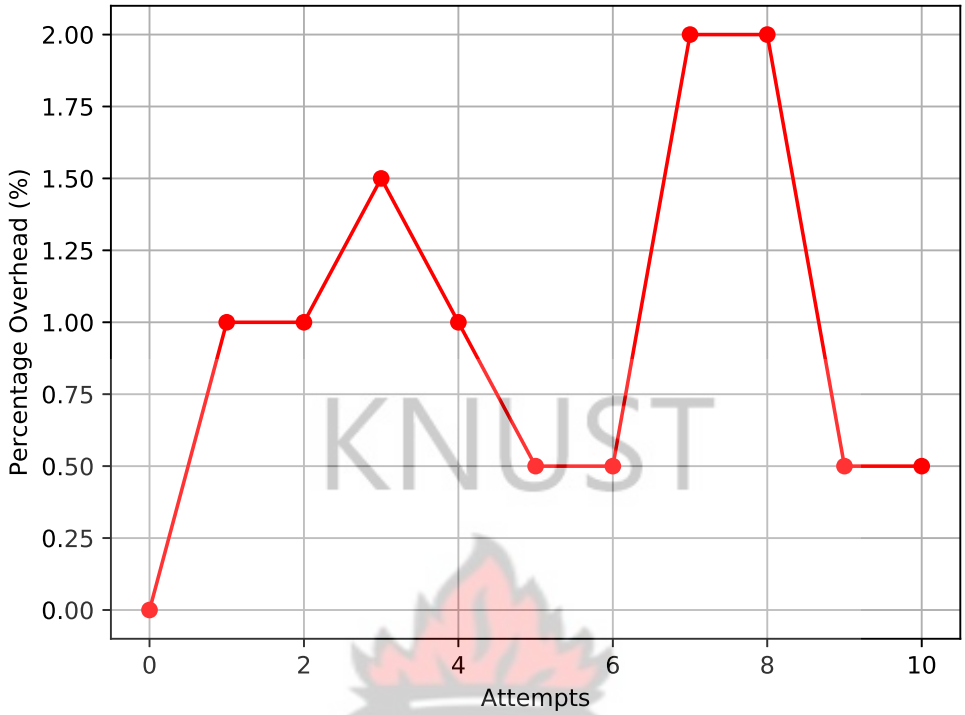


Figure 4.27: Performance Overhead

The average time (shown in Figure 4.28) for detecting MITM attack is **0.1686 seconds**.

KNUST

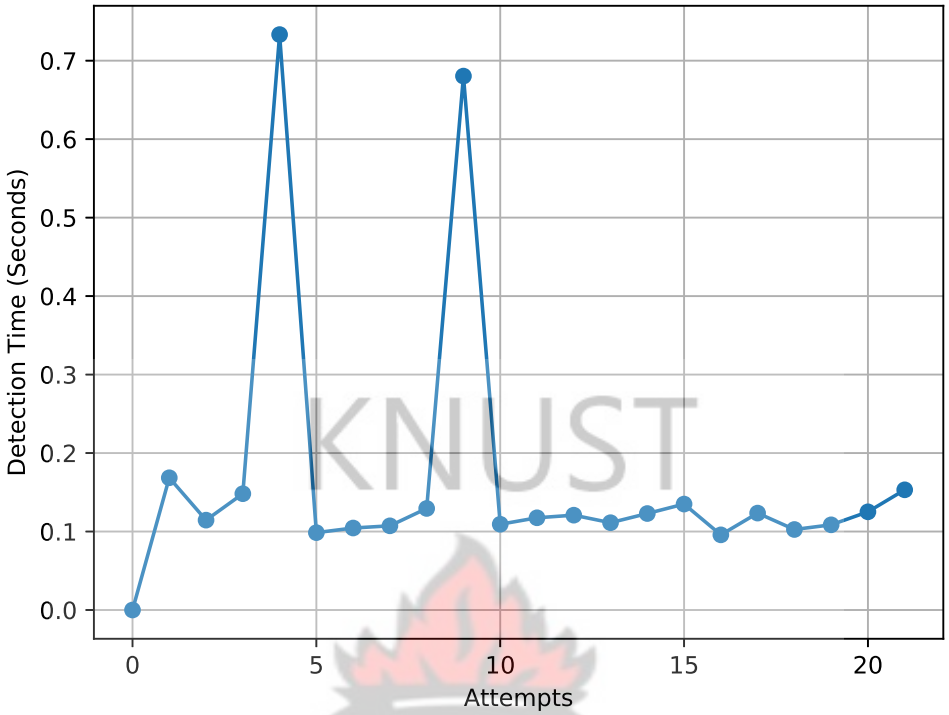


Figure 4.28: Detection Time

The round trip time, which is a measure of latency in a network, for the instance where the algorithm was not implemented is **1.298 seconds** whereas when the algorithm is implemented it is **1.335 seconds**. This shows that the MITM detection and defense algorithm does not affect the latency of the network (shown in Figure 4.29).

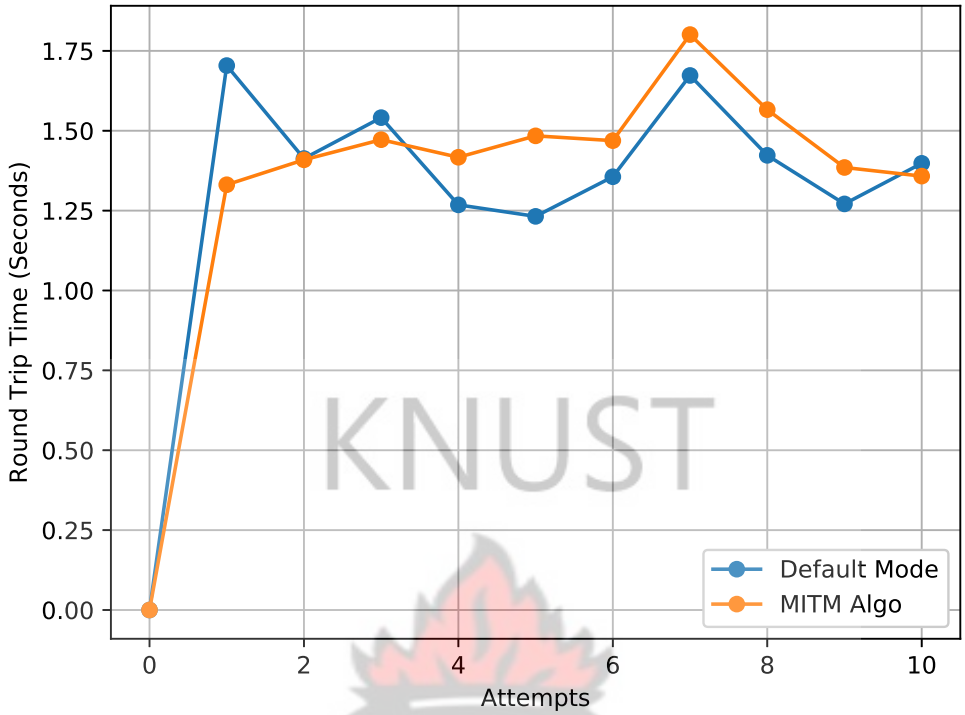


Figure 4.29: Round Trip Time

4.3 Rogue Access Point (RAP) Detection

The detection time together with the CPU usage was measured in all scenarios. The distance between the legitimate AP and the RAP was varied from 1 to 30 meters.

The detection time in Scenario 1 is shown in Fig. 4.30. The detection time averages **2.15064** seconds (shown in Fig. 4.38).

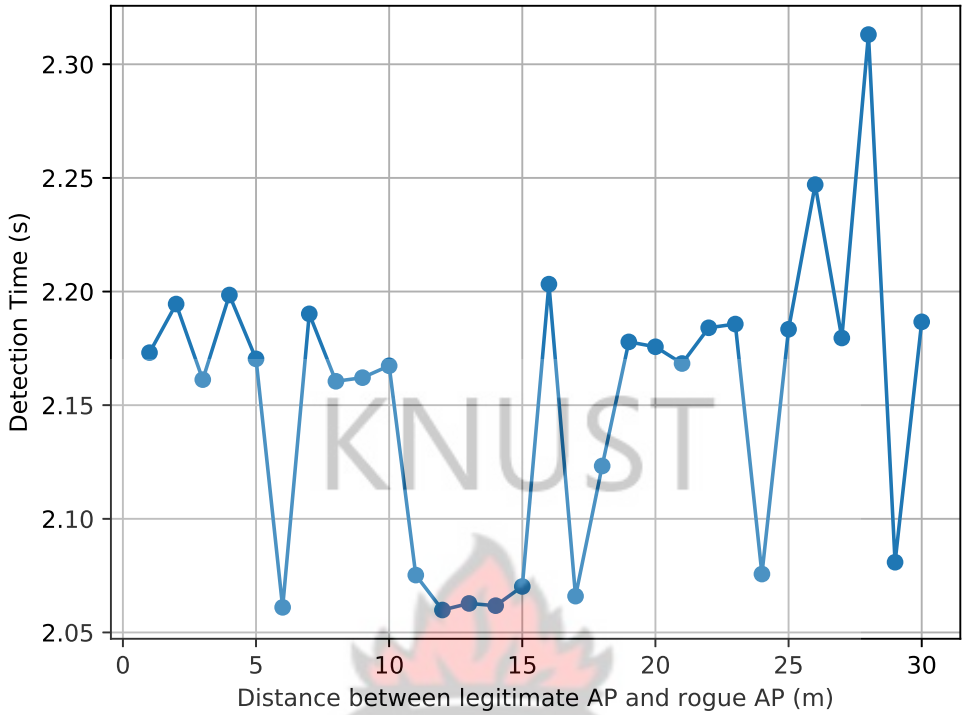


Figure 4.30: Graph showing the detection time using Algorithm 1 (Scenario 1)

The CPU usage of Scenario 1 is shown in Fig 4.31. The CPU usage averages **0.31835%** (Fig 4.38).

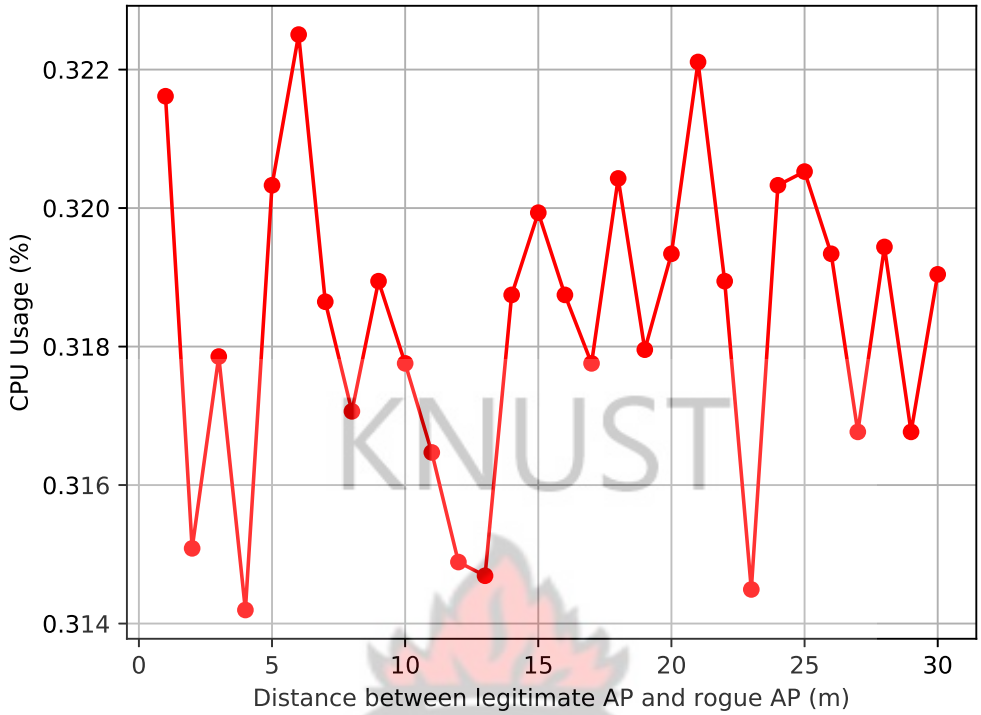


Figure 4.31: Graph showing the CPU usage using Algorithm 1 (Scenario 1)

In Scenario 2, the random channel hoping algorithm is applied after the decoding of the beacon frame in RAP detection. The detection time is shown in Fig 4.32. The detection time averages **0.13182** seconds.

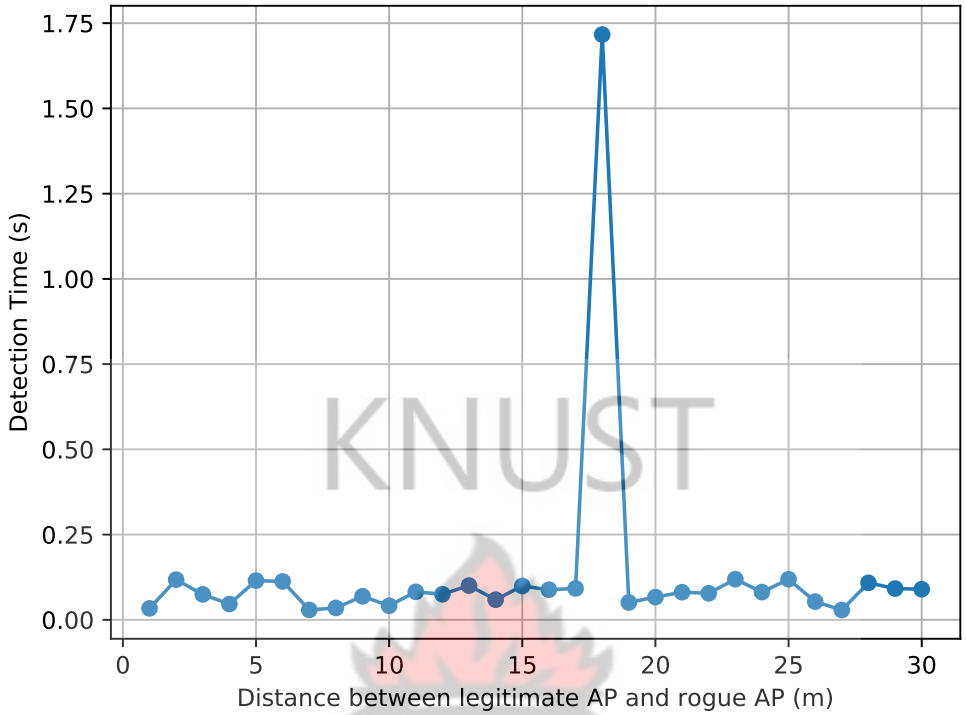


Figure 4.32: Graph showing the detection time using Algorithm 2 with random channel hopping (Scenario 2)

The CPU usage in Scenario 2 (shown in Fig 4.33), averages **2.3049%**(Fig. 4.38).

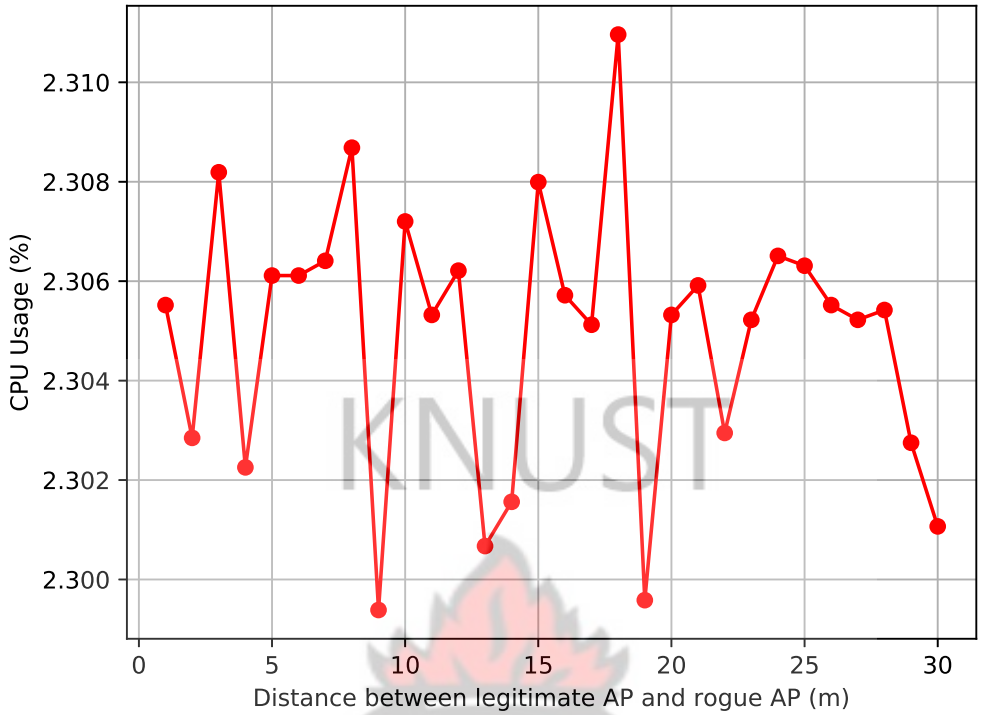


Figure 4.33: Graph showing the CPU usage using Algorithm 2 with random channel hopping (Scenario 2)

Scenario 3 employs iterative channel hopping technique. The detection time (shown in Fig. 4.34) averages **0.20465** seconds.

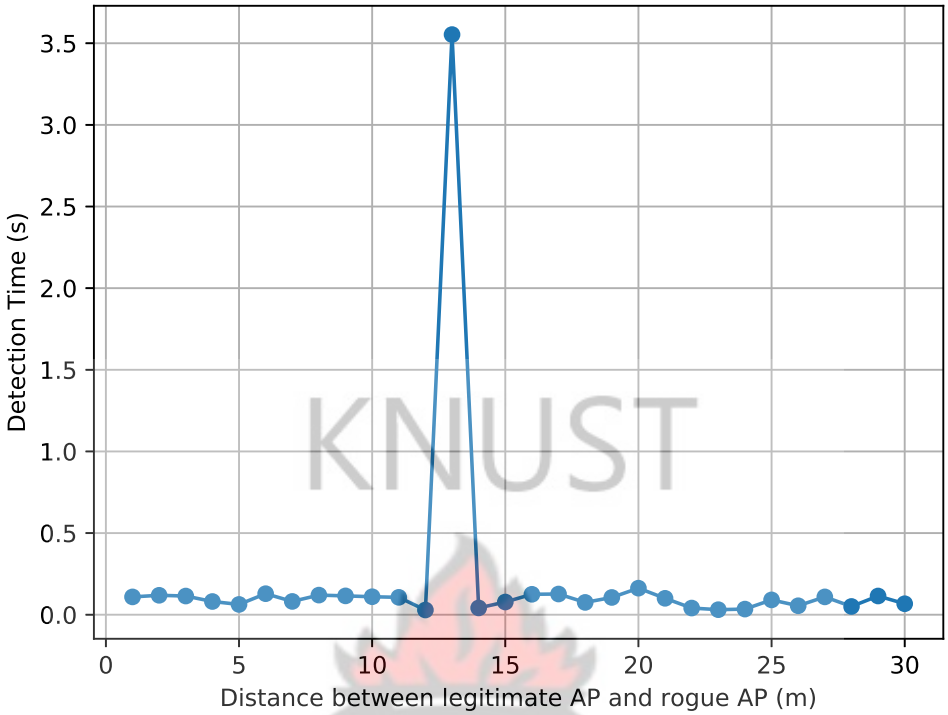


Figure 4.34: Graph showing the detection time using Algorithm 2 with iterative channel hopping (Scenario 3)

The CPU usage if Scenario 3 (shown in Fig. 4.35) averages **2.3051%**.

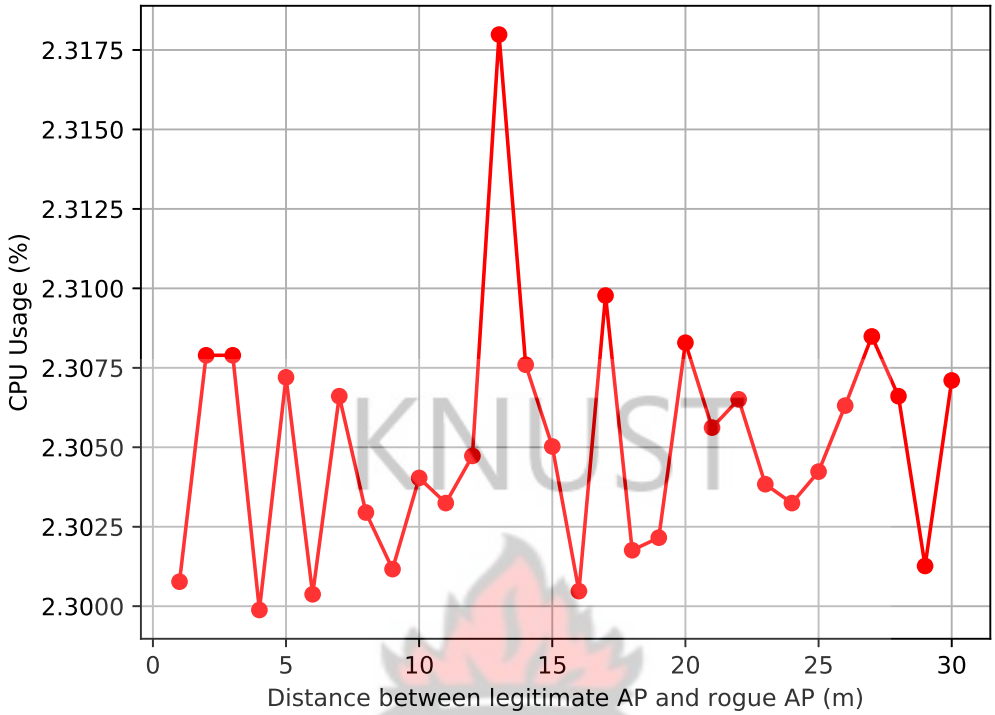


Figure 4.35: Graph showing the CPU usage using Algorithm 2 with iterative channel hopping (Scenario 3)

Fig 4.36 shows the comparison of the detection time of Scenarios 1, 2 and 3 over the measured distance. It can be observed that Scenarios 2 and 3 have a better detection time as compared to Scenario 1.

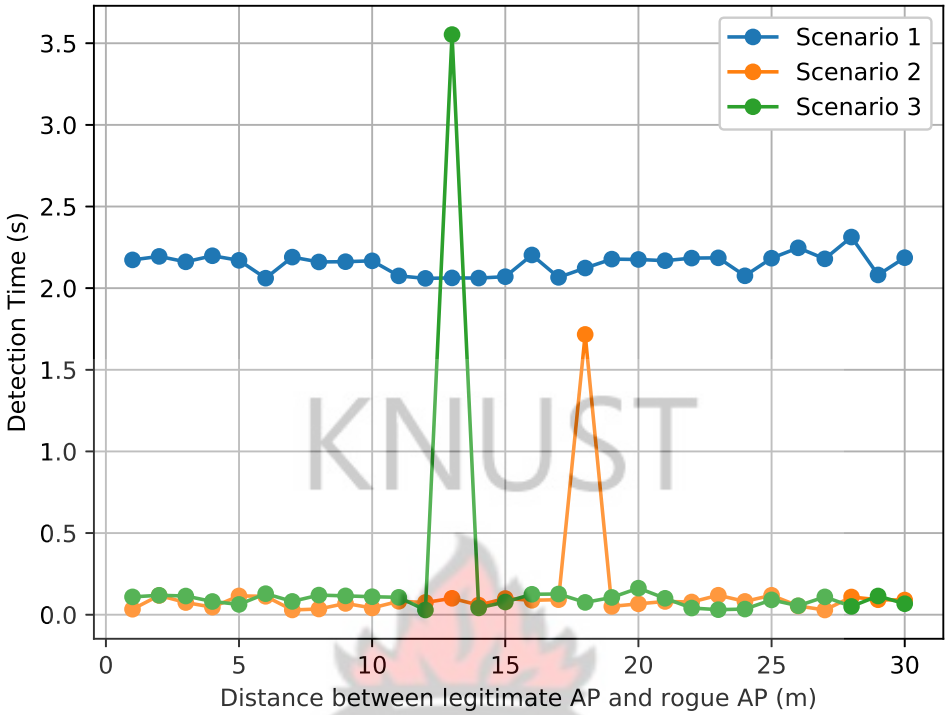


Figure 4.36: Graph showing the comparison of the detection time in all three scenarios

In Fig. 4.37, Scenario 1 performs better with respect to CPU utilization as compared to Scenarios 2 and 3.

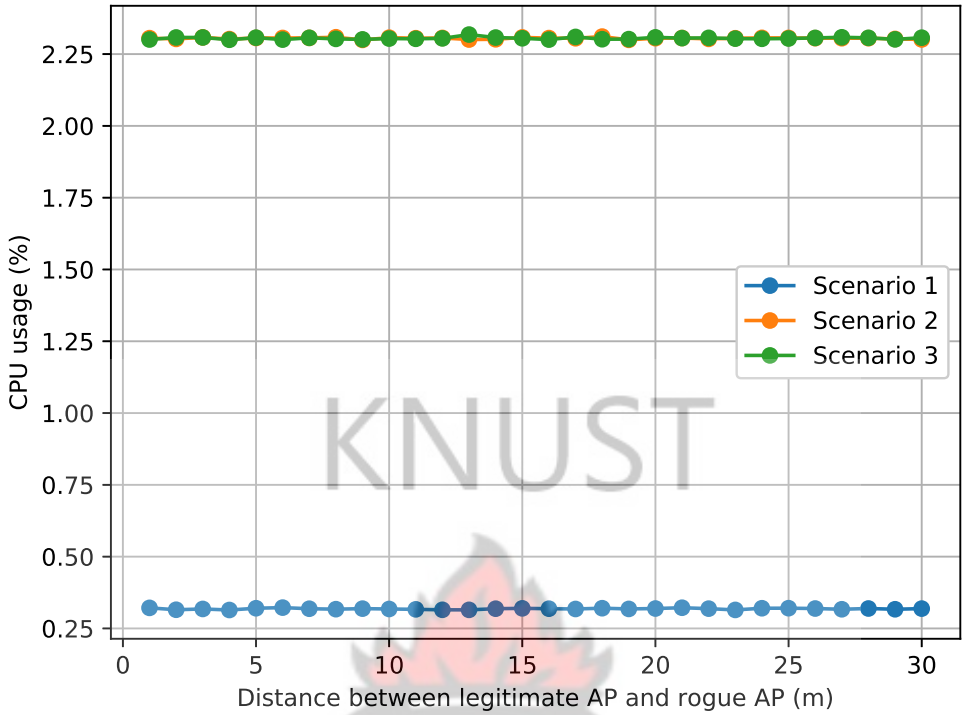


Figure 4.37: Graph showing the comparison of the CPU usage in all three scenarios

In Fig. 4.38, Scenario 2 and 3 outperform Scenario 1 with respect to the detection time. On the other hand, Scenario 1 outperforms Scenarios 2 and 3 in terms of efficient CPU utilization.

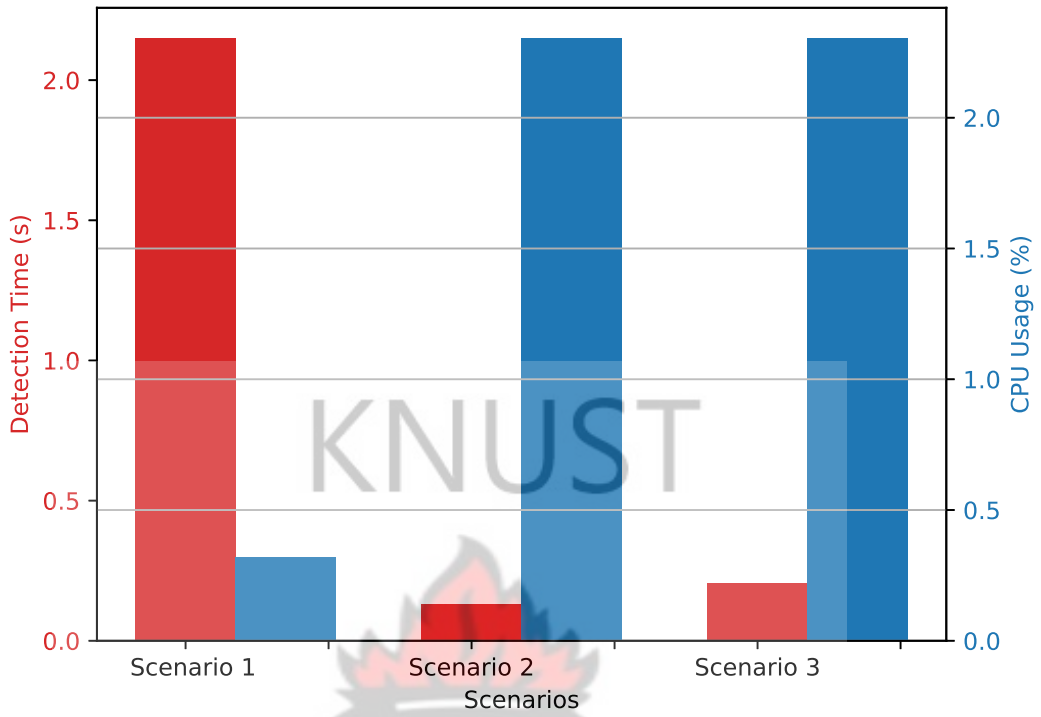


Figure 4.38: Comparison of the average detection time and CPU usage of all three scenarios

From the above results, Algorithm 4 can be implemented on embedded IoT devices that do not transmit data at very short intervals i.e. $intervals > 2.5\,seconds$. For embedded IoT devices that transmit data at very short intervals, Algorithm 5 can be implemented since a CPU usage of 2.3% is quite efficient.

4.4 Orchestration Framework

A benchmark performance on the two algorithms was conducted on a RaspberryPi[45] to determine which of the two is less computationally intensive and what length of bits will be appropriate for use. This is shown in Figure 4.39.

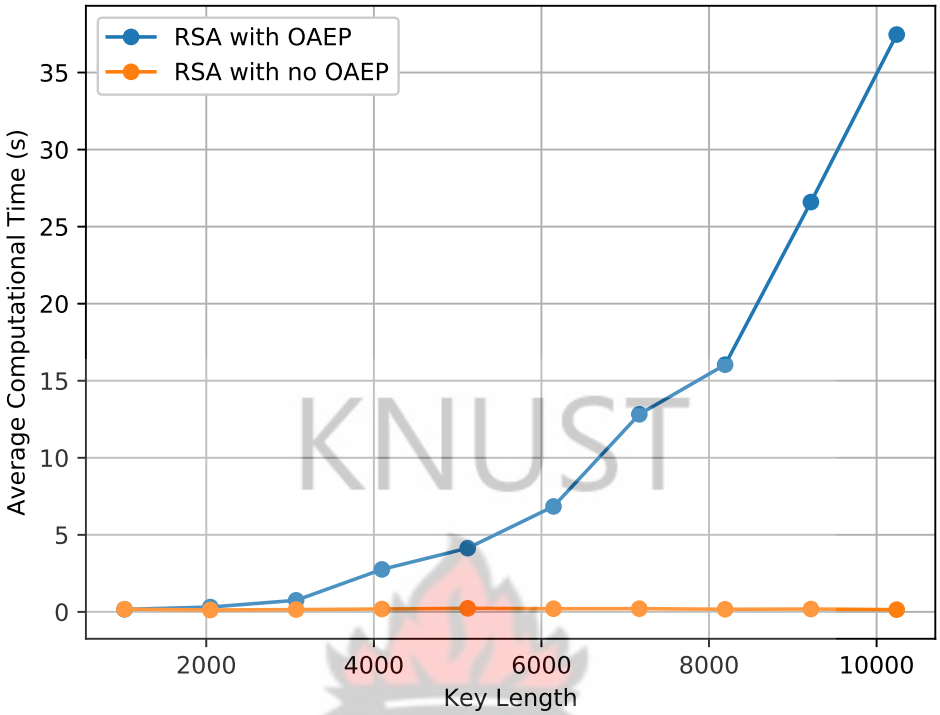


Figure 4.39: Benchmark Performance of RSA and RSA with OAEP

Key lengths from 1024 to 4096 for both RSA and RSA with OAEP produce an average computational time of **0.147 seconds** and **0.408 seconds** as compared to fast Elliptic Curve Digital Signature Algorithm (ECDSA) with an average computational time of **9.085 seconds**[53]. A key length of 2048 was used in the orchestration in the RSA-OAEP algorithm. This offers both low computational overhead together with secure communication process.

The average CPU utilization of the framework implemented on an IoT device averages **0.8%** (shown in Figure 4.40).

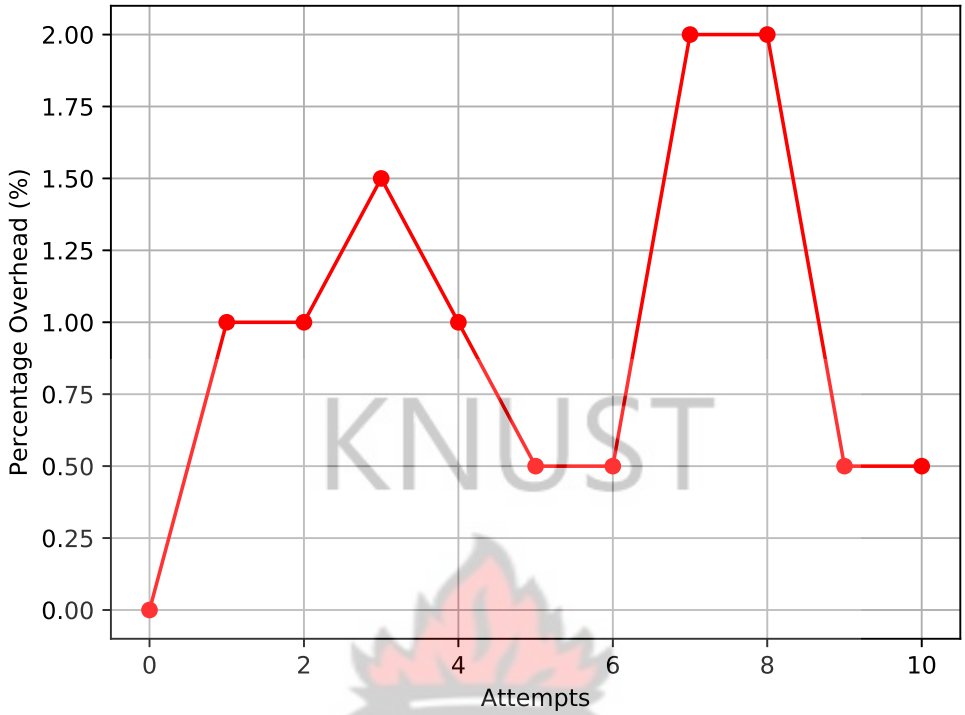


Figure 4.40: CPU Utilization when algorithm was implemented on an IoT Device

Revoking and restoring an IoT device's access was done at the controller (described in Figure 3.12) and the duration was measured. The revocation and restoration time averaged **4.225 seconds** and **4.272 seconds** (shown in Figure).

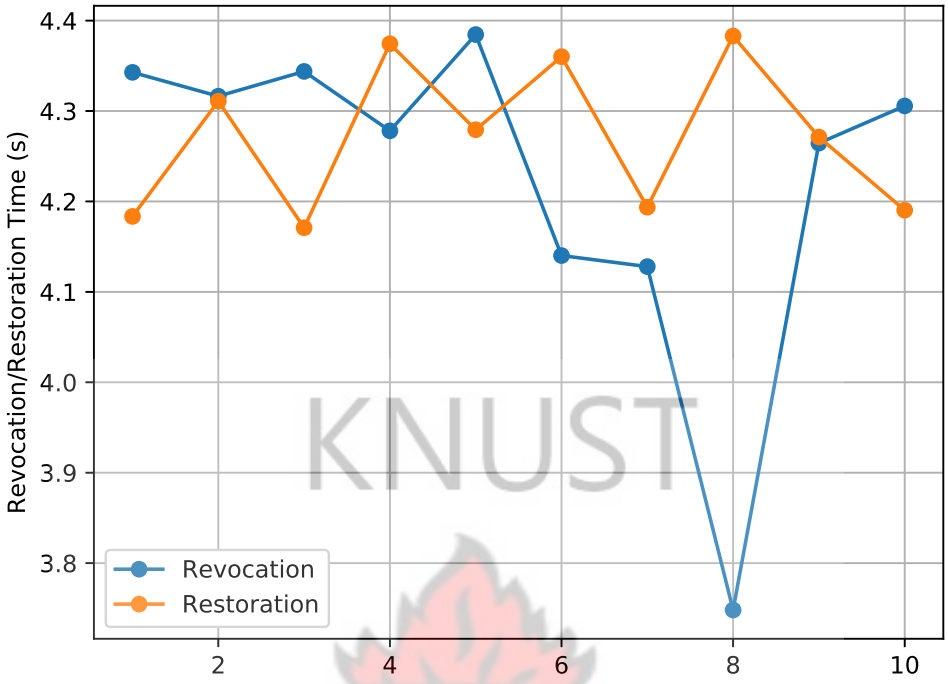


Figure 4.41: Revocation and Restoration Time

KNUST

Chapter 5

CONCLUSION AND RECOMMENDATION

This research has proposed lightweight MITM and RAP Detection algorithms and an orchestration framework. The proposed MITM detection algorithm outperforms the conventional approach suggested by [24] (shown in Table 5.1).

KNUST

Table 5.1: Comparison of MITM Detection Algorithms

KPIs (Avg.)	**Proposed Algorithm**	**Related Works**
CPU Utilization (%)	0.9545	1.65 (Ibrahim et al.)
Detection Rate (Sec.)	0.1686	-
Network Latency	1.298 / 1.335	-

Previous research works do not address RAP detection for IoT devices, hence the proposed RAP detection algorithms. All three scenarios have minimal performance overhead and good detection rates (shown in Table 5.2).

Table 5.2: Comparison of RAP Detection Algorithms

KPIs (Avg.)	**Proposed Algorithm (S1 / S2 / S3)**	**Related Works**
CPU Utilization (%)	0.31835 / 2.3049 / 2.3051	-
Detection Rate (Sec.)	2.15064 / 0.13182 / 0.20465	-

The proposed orchestration framework also has minimal performance overhead and good revocation and restoration time (shown in Table 5.3). This framework can be used by IoT device manufacturers to deliver firmware updates to IoT devices as a means of patching identified vulnerabilities.

Table 5.3: Comparison of Orchestration Framework Algorithms

KPIs (Avg.)	**Proposed Algorithm**	**Related Works**
CPU Utilization (%)	0.8	-
Revocation (Sec.)	4.225	-
Restoration (Sec.)	4.272	-

Future works will consider providing a cross-proxy between the framework

and other existing protocols such as Constrained Application Protocol (CoAP); which is used by resource constrained IoT devices.

It is recommended that a lightweight Denial of Service (DoS) detection and defense mechanism be implemented for resource-constrained IoT devices. Intrusion detection systems that address other technologies such as Bluetooth Low Energy (BLE), Z-Wave and Near-Field Communication (NFC) should be addressed. The entire source code for the research work can be found at [54].

KNUST

REFERENCES

[1] I. Lee and K. Lee, *The internet of things (iot): Applications, investments, and challenges for enterprises*, Business Horizons, vol. 58, no. 4, pp. 431-440, 2015.

[2] Dave Evans, *The Internet of Things: How the Next Evolution of the Internet Is Changing Everything*, Cisco Internet Business Solutions Group (IBSG), pp. 3, April 2011.

[3] Dave Evans, "The Internet of Things: How the Next Evolution of the Internet Is Changing Everything" Cisco Internet Business Solutions Group (IBSG) , pp. 4, April 2011.

[4] D. Miorandi , S. Sicari , F. De Pellegrini, I. Chalmatac, *Internet of Things: vision, applications and research challenges*, Ad Hoc Network 10(7), 1497 - 1516, 2012.

[5] E. Borgia, *The Internet of Things vision: key features, applications and open issues*. Comput. Commun. 54, 1-31, 2014

[6] Garcia-Morchon O., Kumar S., Struik R., Keoh S., Hummen R., *Security considerations in the IP-based Internet of Things*, IETF Internet-Draft, 2013.

[7] Manos Antonakakis, Tim April et al, "Understanding the Mirai Botnet", 26th USENIX Security Symposium, pp. 1, August 2017.

[8] Selena Larson, "FDA confirms that St. Jude's cardiac devices can be hacked",*http://money.cnn.com/2017/01/09/technology/fda-st-jude-cardiac-hack/*, [Accessed Online], 8th July, 2018.

[9] Lain Thompson, "Wi-Fi baby heart monitor may have the worst IoT security of 2016", www.theregister.co.uk/2016/10/13/possibly_worst_iot_security _failure_yet/?mt=1476453928163, [Accessed Online] 8th June, 2018.

KNUST

[10] Sicari S., Rizzardi A., Grieco L., Coen-Porisini A., *Security, privacy and trust in Internet of Things: the road ahead*, Comput. Netw. 76 (0), 146-164, 2015.

[11] Notra S., Siddiqi M., Gharakheili H., Sivaraman V., Boreli R., *An experimental study of security and privacy risks with emerging household appliances*. In: Communications and Network Security (CNS), 2014 IEEE Conference on, pp. 79-84, 2014.

[12] Kolias, C., Stavrou, A., Voas, J., Bojanova, I., Kuhn, R., *Learning Internet-of-things security "Hands-on"*. IEEE Secur. Priv. 20 (February), 2-11. http://dx.doi.org/10.1109/MSP.2016.4, (January/February).

[13] Plummer, D.C. (1982) An Ethernet Address Resolution Protocol. RFC 826.

[14] AI Sukkar G. , Saifan R., Khwaldeh S., Maqableh M., Jafar I., *Address Resolution Protocol (ARP); Spoofing Attack and Proposed Defense*, Communications and Network, 8, 118-130, 2016.

[15] Mauro Conti, Nicola Dragoni, Viktor Lesyk, *A Survey of Man In The Middle Attacks*, IEEE Communications Surveys & Tutorials, Vol. 18, No. 3, 2016.

[16] J. Belenguer, C.T. Calafate, *A low-cost embedded IDS to monitor and prevent man-in-the-middle attacks on wired LAN environments*, Proc. Int. Conf. SecureWave Emerging Secur. Inf. Syst. Technol., 2007, pp. 122-127.

[17] Issac B., *Secure ARP and Secure DHCP Protocols to Mitigate Security Attacks*, International Journal of Network Security, 8, 107-118, 2009.

[18] D. Bruschi, A. Ornaghi, E. Rosti, *S-ARP: A secure address resolution protocol*, Proc. 19th Annu. Comput. Secur. Appl. Conf., pp. 66-74, 2003.

[19] Lootah W., Enck W., McDaniel P, *TARP: Ticket-Based Address Resolution Protocol*, Computer Networks, 51, 4322-4337, 2007.

KNUST

[20] R Philip, *Securing Wireless Networks from ARP Cache Poisoning*, Master's Thesis, San Jose State University, (2007).

[21] S. Y. Nam, D. Kim, J. Kim, *Enhanced ARP: Preventing ARP poisoning-based man-in-the-middle attacks*, IEEE Commun. Lett., vol. 14, No. 2. pp. 187-189, 2010.

[22] S. Y. Nam, S. Jurayev, S.-S. Kim, K. Choi, G. S. Choi, *Mitigating ARP Poisoning-Based Man-In-The-Middle Attacks in Wired or Wireless LAN*

[23] S. Y. Nam, S. Djuraev, M. Park, *Collaborative Approach to Mitigate ARP Poisoning-Based Man-In-The-Middle Attack*, Comput. Netw. vol 57, No. 18. pp 3866-3884, 2013.

[24] Ibrahim Halil Saruhan, *Detecting and Preventing Rogue Devices on the Network*, SANS Institute, pp. 5-7 2007.

[25] S.B.Vanjale, Amol K. Kadam, Pramod A. Jadhav, *Detecting & Eliminating Rogue Access Point in IEEE 802.11 WLAN*, International Journal of Smart Sensors and Ad Hoc Networks (IJSSAN) Volume-1, Issue-1, 2011.

[26] T. Kim, H. Park, H. Jung, H. Lee, *Online detection of fake access points using received signal strengths*, 2012.

[27] Mehndi Samra, Mehak Mengi, Sparsh Sharma, Naveen Kumar Gondhi, *Detection and Mitigation of Rogue Access Point*, Journal of Scientific and Technical Advancements, Volume 1, Issue 3, pp. 195-198, 2015.

[28] Ettercap, http://ettercap.github.io/ettercap/ [Accessed Online], January 2019.

[29] Wireshark, http://www.wireshark.org/ [Accessed Online], January 2019

[30] Snort, http://www.snort.org/ [Accessed Online], January 2019.

KNUST

[31] Amazon, "Aws iot framework", *https://aws.amazon.com/iot*, [Accessed Online], January 2019.

[32] Mahmound Ammar, Giovanni Russello, Bruno Crispo, "Internet of Things, A Survey on the Security of IoT Frameworks", Journal of Information Security and Applications, pp. 8-27, 2018.

[33] ARM, "Arm mbed iot device platform", *http://www.arm.com/products/iot-solutions/mbed-iot-device-platform* , [Accessed Online], January 2019.

[34] Microsoft, "Tap into the internet of your things with azure iot suite", *https://www.microsoft.com/en-us/cloud-platform/internet-of-things-azure-iot-suite*, [Accessed Online], January 2019.

[35] MSV J. , "Google brillo vs. apple homekit: The battleground shifts to iot", *https://www.forbes.com/sites/janakirammsv/2015/10/29/google-brillo-vs-apple-homekit-the-battleground-shifts-to-iot/*, [Accessed Online], January 2019.

[36] Ericsson, "Open source release of iot app environment calvin", *https://www.ericsson.com/en/blog/2015/6/open-source-release-of-iot-app-environment-calvin*, [Accessed Online], January 2019.

[37] Apple, "The smart home just got smarter", *http://www.apple.com/ios/home/*, [Accessed Online], January 2019.

[38] Organization E., "Kura framework", *http://www.eclipse.org/kura/*, [Accessed Online], January 2019.

[39] Zhenyu Wen, Renyu Yang, Peter Garraghan, Tao Lin, Jie Xu, and Michael Rovatsos, *Fog Orchestration for IoT Services: Issues, Challenges and Directions*, pp. 1, IEEE Internet Computing - March 2017.

[40] Alejandro G., Manuel A. A., Jordan P. E., Oscar S. M., Juan M. C. L., Cristina P. G-B., *Introduction to Devices Orchestration in Internet of Things*

Using SBPMN , International Journal of Interactive Multimedia and Artificial Intelligence , December 2011.

[41] L. Galluccio, S. Milardo, G. Morabito and S. Palazzo, *SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks*, IEEE Conference on Computer Communications (INFOCOM) (2015), 513-521.

[42] D. Gante, M. Aslan and A. Matrawy, *Smart wireless sensor network management based on software-defined networking*, 27th Biennial Symposium on Communications (QBSC) (2014), 71-75.

[43] T. Miyazaki, S. Yamaguch, K. Kobayashi, J. Kitamichi, S. Guo, T. Tsukahara and T. Hayashi , *A software defined wireless sensor network*, International Conference on Computing, Networking and Communications (ICNC) (2014), 847-852.

[44] Bruno Bogaz Zarpelao, Rodrigo Sanches Miani, Caludio Toshio Kawakani, Sean Carlisto de Alvarenga, *A Survey of Intrusion Detection in Internet of Things*, Journal of Network and Computer Applications, pp. 2-4, 2017.

[45] Raspberry Pi 3 Model B, https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ [Accessed Online], January 2019.

[46] NodeMcu, "Connect things easy", http://www.nodemcu.com/index_en.html, [Accessed Online], January 2019.

[47] Jay Lux Ferro, "Iwlist Parser", https://github.com/jayluxferro/iwlist-parser [Accessed Online] April 2019.

[48] Iwlist, https://linux.die.net/man/8/iwlist [Accessed Online], January 2019.

[49] Mangle, "Drivers for the rtl8192eu chipset for wireless adapters ", https://github.com/Mange/rtl8192eu-linux-driver/ [Accessed Online], January 2019.

KNUST

[50] *Digital Signature Standard*, Information Technology Laboratory, National Institute of Standards and Technology, pp. 9, July 2013.

[51] OpenWrt, *https://openwrt.org*, [Accessed Online] 12th July, 2018.

[52] pfSense, *https://pfsense.org*, [Accessed Online] 13th July, 2018.

[53] Anton Kueltz, "Python library for fast elliptic curve crypto", https://github.com/AntonKueltz/fastecdsa [Accessed Online], 12th January 2019.

[54] Jay Lux Ferro, "Intrusion Detection System for IoT Devices", https://github.com/jayluxferro/IoT-IDS/, [Accessed Online], April 2019.

KNUST

Appendix

Research Publications

1. Justice Owusu Agyemang, Jerry John Kponyo, and Griffith Selorm Klogo, "The State of Wireless Routers as Gateways for Internet of Things (IoT) Devices." Information Security and Computer Fraud, vol. 6, no. 1 (2018): 8-18. doi: 10.12691/iscf-6-1-2.

2. Justice Owusu Agyemang, Jerry John Kponyo, and Isaac Acquah, "Lightweight Man-In-The-Middle (MITM) Detection and Defense Algorithm for WiFi-Enabled Internet of Things (IoT) Gateways." Information Security and Computer Fraud, vol. 7, no. 1 (2019): 1-6. doi: 10.12691/iscf-7-1-1.

3. Justice Owusu Agyemang, Jerry John Kponyo, and Griffith Selorm Klogo, "A Lightweight Rogue Access Point Detection Algorithm for Embedded Internet of Things (IoT) Devices." Information Security and Computer Fraud, vol. 7, no. 1 (2019): 7-12. doi: 10.12691/iscf-7-1-2.

4. Justice Owusu Agyemang, Jerry John Kponyo, "An Orchestration Framework for IoT Devices based on Public Key Infrastructure (PKI)", Conference: International Journal of Simulation Systems, Science & Technology, March 2019: DOI 10.5013/IJSSST.a.20.S1.04.

KNUST