VULNERABILITY DETECTION AND RESOLUTION IN INTERNET OF THINGS (IoT) DEVICES

Ruth M. Ogunnaike

A Thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in Computer Science and Software Engineering

University of Washington

2017

Committee:

Brent Lagesse, Chair

Marc Dupuis

Yang Peng

Program Authorized to Offer Degree:
Computing and Software Systems

University of Washington

Abstract

VULNERABILITY DETECTION AND RESOLUTION IN INTERNET OF THINGS (IoT) DEVICES

Ruth M. Ogunnaike

Chair of the Supervisory Committee:
Professor Brent Lagesse
Computing and Software Systems

The use of Internet of Things (IoT) devices has grown significantly in the past decade. While IoT is expected to improve life for many by enabling smart living spaces, the number of security risks that consumers and businesses face is also increasing. A high number of vulnerable IoT devices are prone to attacks and easy exploit. Previous researches has focused on security that must be implemented by administrators and manufacturers to be effective. This research propose a system that does not rely on best practices by IoT device companies, but rather allows inexperienced users to be confident about the security of the devices that they add to their network.
This research presents an implementation of an IoT architectural framework, called SeeSec, based on Software Defined Networking (SDN) paradigm. In this architectural framework, IoT devices attempting to join an IoT network are scanned for vulnerabilities using custom vulnerability scanners and penetration testing tools before allowing communication with other devices in the network.

In the case that a vulnerability is detected, the system will try to fix the vulnerability. If the fix fails, the user will be alerted to the vulnerability and provided with suggestions for fixing it before the device will be allowed to join the network.

SeeSec is built on both existing corporate vulnerability scanners, and custom scanners to run security scans in the IoT devices. The research aim to build a user friendly

system that makes it easy for non-technical users to understand the suggestions provided by the system to resolve detected vulnerabilities that the system is unable to resolve automatically.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

VULNERABILITY: A flaw in code that resides in IoT devices or design of IoT devices that creates a potential point of security compromise in an IoT network.

INTERNET OF THINGS (IOT): Collection of interconnected uniquely identifiable embedded computing devices that communicates to provide data and services for applications that drive smart living spaces.

SOFTWARE DEFINED NETWORKING (SDN): A computer networking approach that enables network administrators to programmatically initialize, control, change, and manage network behavior dynamically via open interfaces and abstraction of lower-level functionality. It separates the control plane and data plane in a traditional network architecture.

SDN CONTROLLER: An application in software defined networking that manages flow control to enable intelligent networking. SDN controllers are based on protocols that allow servers to tell switches where to send packets.

OPENFLOW: A standard communication interface defined between the control and forwarding layers of an SDN architecture.

VULNERABILITY SCANNER: A computer program designed to assess computing devices, computer systems, networks or applications for weaknesses. It can also be defined as a tool used for identifying possible threat agents in an IoT device.

DISTRIBUTED DENIAL OF SERVICE (DDOS): A cyber attack in which multiple compromised devices attack a target, such as server, website or other network

resource, and cause a denial of service for legitimate users of the target resource.

BLACKLIST: A list of unique identifiers of IoT devices that SeeSec flagged as vulnerable.

WHITELIST: A list of unique identifiers of IoT devices that SeeSec flagged as non-vulnerable.

ACKNOWLEDGMENTS

I want to express my sincere appreciation to my committee members who were generous with their expertise and time, as well as providing resources needed to successfully implement this research. Special thanks to my committee chair, Dr. Brent Lagesse for his invaluable assistance, support, and effort towards the success of this research. Thanks to Dr. Marc Dupuis, and Dr. Yang Peng for their time and support.

I would also like to acknowledge and thank the individuals who participated in the research's system evaluation experiment for their time and valuable feedback. Thanks to my graduate advisor, Ms. Megan Jewell for her help and assistance throughout the degree program. Thank you to my family and friends for their moral support, and special thanks to my sister, Busola Ogunnaike for her words of encouragement and for always being there for me throughout the entire masters program.

Finally, I wish to express my since appreciation to University of Washington Bothell, and others who have helped with this research.

DEDICATION

to my parents, Emmanuel and Elizabeth Ogunnaike.

Chapter 1

INTRODUCTION

Internet of Things (IoT) is a collection of interconnected embedded computing devices, objects, and services that can communicate and share data/information to achieve a common goal in different areas and application infrastructure [2]. These applications use many sensors, actuators, and routers which connect to the Internet through wired or wireless networks called IoT devices [3].

Each IoT device has a unique identifier and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction [4], which can pose risks to human life.

The use of IoT devices is growing rapidly in the last decades and continues to develop in terms of dimension and complexity. At the end of 2014, 42.3% of the world population was connected to the network [5]. The security threats and concerns increases with the evolution of the Internet of Things (IoT), since it will include every object or device with networking capabilities. The number of Internet-connected devices is expected to reach 35 billion by 2020 [1]; a majority of these devices being IoT and wearable devices and a higher percentage in home automation (Figure 1.1).

The Internet of Things is becoming an integral part of our daily lives in the form of wearable computers, smart health trackers, connected smoke detectors, light bulbs, baby monitors, digital video recorders, smart refrigerators, CCTV video cameras, and essentially any other Internet-connected device. These IoT devices are armed with an array of sensors that enable heterogeneous wireless communication. It is crucial that these devices undergo thorough testing and establish minimum baseline for security. However, many devices have already been deployed insecurely as demonstrated by a number of recent distributed denial of service (DDoS) attacks that leverage vulnerable IoT devices [6]. In addition, the hardware (chipset) mostly used in the IoT devices are

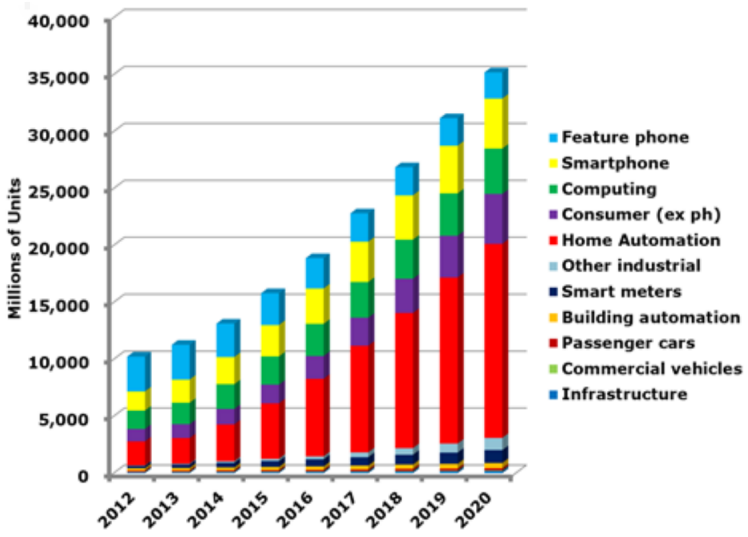

Figure 1.1: Total Internet of Things (IoT) Connected Devices, Installed Base in millions [1].

old and often have multiple known vulnerabilities.

1.1 Motivation

Traditional security mechanisms like firewalls, intrusion detection, and prevention systems are deployed at the Internet edge [5]. Those mechanisms are used to protect the network from external attacks; such mechanisms are no longer enough to secure the next generation Internet. The borderless architecture of the IoT raises additional concerns over network access control and software verification.

In addition, the firmware and software installed on these IoT devices rarely gets any in-depth security testing and always has its own set of security issues. This is because the software has focused primarily on functionality, and has not undergone significant security review [7]. Many manufacturers currently fail to implement well-established security standards correctly or emerging techniques at all. Furthermore, IoT devices are often sold with old and unpatched embedded operating systems and firmware, and purchasers often fail to change the default passwords on smart devices or fail to use

sufficiently strong passwords.

A recent distributed denial of service (DDoS) attack was launched on a security news website, KrebsOnSecurity, using vulnerable IoT devices with default settings (factory default credentials). The vulnerable IoT devices were used to launch a massive volley traffic on the website causing the website to be unavailable for days. A similar attack was the DDoS attack on Dyn, a company that hosts domain name servers that help users connect to websites. The DDoS attack prevented users from accessing numerous large websites and services, including Twitter and Paypal that uses hosting service of Dyn.

Recent advances in computer networking have introduced a new technology paradigm for communication, the Software Defined Networking (SDN); a central software program, called SDN controller, manages the overall network behaviour. Based on the SDN architecture, we propose a security framework for the IoT. The proposed security framework, SeeSec was designed to establish and secure both wired and wireless network infrastructure. To enhance security in the IoT network, IoT devices are scanned for vulnerabilities when they are first added to the network. The research also aim to automatically resolve detected vulnerabilities in IoT devices, and provide suggestions on how to fix unresolved vulnerabilities to users.

While it is critical to engineer secure systems [8], the focus of this research is to make it easier for consumers to deploy their IoT devices without having to worry about their devices being made part of an attack or revealing personal information about them due to negligence by the manufacturer.

The vulnerabilities existing in IoT devices and how the security framework mitigate against them is further discussed in this document. In addition, the system design and implementation of the proposed security framework and how it improves IoT security while utilizing SDN architecture and custom vulnerability scanners/penetration testing tools is also discussed in detail in chapter 4 and 5.

This research introduces a novel approach in addressing security issues in the Internet of things. The research enhance security in IoT by preventing vulnerable IoT devices from having communication access in a network. IoT devices are scanned for

vulnerabilities on an attempt to join the network, and detected vulnerabilities are resolved when possible i.e. SeeSec has the resources to resolve a vulnerability, and the user grants SeeSec permission to resolve detected vulnerabilities.

1.2 Major Contributions

SeeSec contributes to IoT security as follows;

1. **Enables non-technical (and technical) users of IoT to securely add IoT devices to their network**. SeeSec detects vulnerabilities in an IoT device and automatically fix detected vulnerabilities. Suggestions to fix unresolved vulnerabilities is provided in a way it is easy for non-technical users to understand. While the focus of this research is to enable non-technical users setup a secured IoT network, SeeSec can also be used by corporate organization network administrators to secure and mitigate against vulnerable devices.

2. **Provides remote and locally deployable security service**. SeeSec can be deployed both locally and remotely; SeeSec can provide a remote security service to multiple network administrators whereby OpenFlow-enabled switches can connect remotely to SeeSec's OpenFlow controller.

3. **Secure and mitigate against vulnerable devices**. SeeSec blacklist vulnerable devices and disable all communication access to and from a vulnerable device as depicted in Figure 1.2.

4. **Creates a testing platform to test for baseline security** The security framework can be used by manufactures to carry out security tests on manufactured IoT devices before distributing the products to consumers.

In addition, SeeSec aim to provide a baseline for further implementation of security programs. The architectural design of SeeSec is scalable, and enhancing efficiency of the primary functions of the system is easy. Multiple vulnerability scanners (both

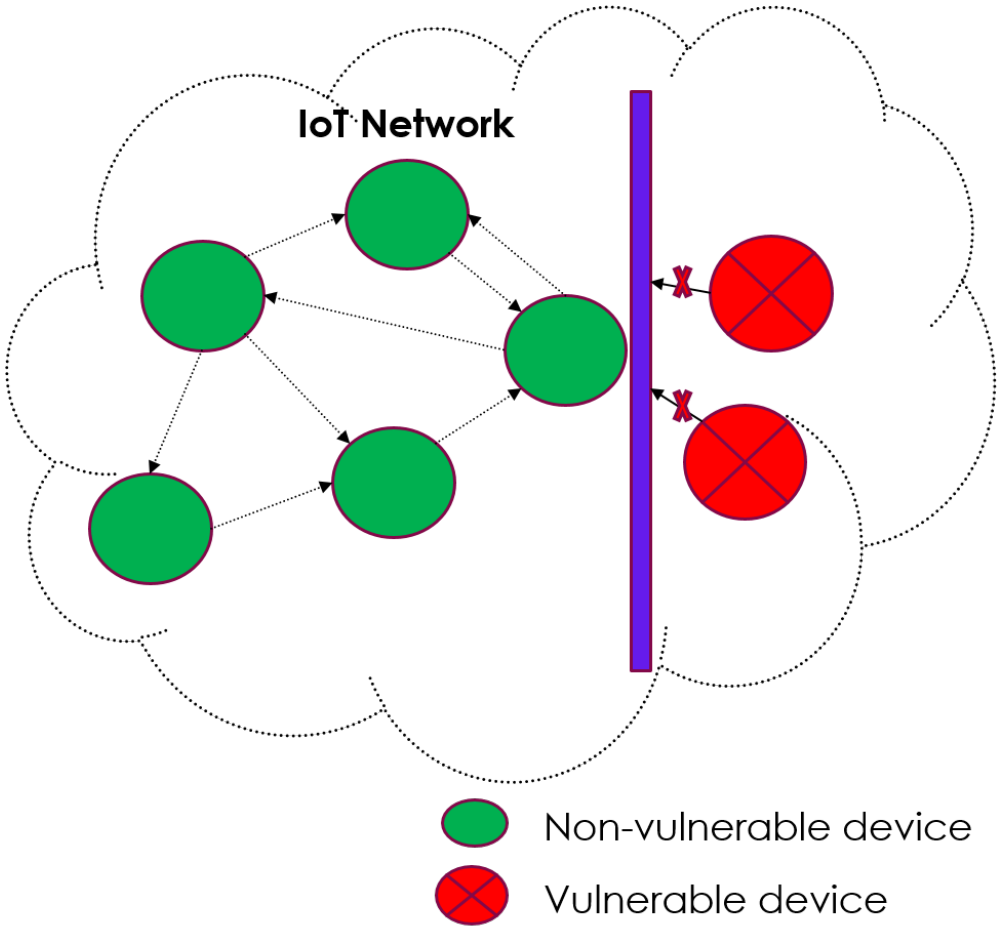

Figure 1.2: SeeSec Blocking Communication Access to and from Vulnerable Devices in an IoT Network

professional penetration testing tools and custom scanners) can be integrated with the system. This ensures continuous improvement of the primary functionalities of the system, by detecting more IoT-specific vulnerabilities.

1.3 Thesis Overview

This research presents a practical, scalable and SDN-based IoT architecture security framework, SeeSec, that scans for vulnerabilities in an IoT device on an attempt to join an IoT network, automatically resolves detected vulnerabilities on behalf of the user when possible (SeeSec can also be configured to automatically resolve vulnerabilities without user's intervention), and provides suggestions to fix unresolved vulnerabilities in a user friendly manner. Given that previous research has attempted to enhance security in the Internet of things by implementing software security programs in specific layers of the IoT architecture, we attempt to enhance security in the Internet of Things via early detection and resolution of vulnerabilities in IoT devices.

This document is organized as follows. Chapter 2 discusses current traditional IoT architecture, IoT security and privacy challenges, and vulnerabilities in IoT devices. Chapter 3 discusses work that has been done towards enhancing IoT security. Categories of security mechanisms proposed by previous works that was discussed in this document include; authentication measures, trust mechanism, federated architecture, SDN-based architecture, and existing IoT security services. We highlight the areas where our research differs from the state of art and the contributions made.

Chapter 4 discusses the design goals of the proposed system, our security model and assumptions, proposed IoT network architecture, and process flow of SeeSec. Chapter 5 presents the implementation of the core components (SDN controller, Scan server, and Vulnerability scanners) of the proposed system, SeeSec.

In chapter 6, we present results from the experiments carried out to evaluate the performance and efficiency of SeeSec based on the defined metrics. In addition, the methodology and experimental set up is discussed in this chapter. Chapter 7 concludes by highlighting the gaols achieved and the areas that require further research, and suggests improvements that can be done in the future work.

Chapter 2

BACKGROUND

2.1 IoT Architecture, and Security Challenges

The traditional security goals of Confidentiality, Integrity and Availability (CIA) also apply to IoT. However, IoT has resource limitations in terms of device components (memory), and computational power. In securing the IoT, it is important to ensure accuracy and validity of data exchanged on the network, and ensure data is available to users when needed. The general IoT architecture consists of three layers: the perception layer, the network layer, and the application layer (Figure 2.1). Each layer is defined by its functions, and there are security issues specific to each layer.

2.1.1 Perception layer

The perception layer is also known as the physical layer [9] or the sensors layer in IoT [2]. The purpose of this layer is to acquire data from the environment with the using of sensors and actuators. This layer detects, collects and process information, and then transmit the information to the network layer.

The confidentiality of this layer can easily be exploited by replay attacks [2] which can be made by spoofing, altering or replaying the identity information of one of the IoT devices. In addition, attackers might also gain an encryption key by analyzing the required time to perform encryption.

2.1.2 Network layer

The network layer of IoT serves the function of data routing and transmission to different IoT hubs and devices over the Internet. Cloud computing platforms, Internet gateways, writing and routing devices, are on this layer; and they operate by using recent communication technologies such as WiFi, and Bluetooth. The network gate-

ways serve as the mediator between different IoT nodes by aggregating, filtering, and transmitting data to and from different sensors.

The network layer is highly susceptible to Man-in-the-Middle attack, which can be followed by eavesdropping. If the keying material of the devices is eavesdropped, the secure communication channel can be completely compromised. In addition, a hacker can attack the confidentiality and privacy at network layer by traffic analysis, and passive monitoring due to the frequent occurrence of remote access mechanisms and data exchange in IoT devices.

2.1.3 Application Layer

The application layer guarantees the authenticity, integrity, and confidentiality of data in IoT. This layer is service-oriented [9] which ensures the same type of services among the connected devices. At this layer, the purpose of IoT or the creation of a smart environment is achieved.

The Internet of things does not have global policies and standard that govern the interaction and development of applications, hence different applications adopt different authentication mechanisms. This makes integration of all of the devices difficult to ensure data privacy and identity authentication. The lack of policies and standard governance is a contributing factor to the increasing number of vulnerabilities found in IoT devices.

2.2 Vulnerabilities in IoT

New vulnerabilities in IoT devices keep emerging as its use increases. According to OWASP (Open Web Application Security Project), an online community that creates freely-available articles, methodologies, documentation, tools, and technologies in the field of web application security [10], the top 10 collection of potential risks and vulnerabilities of Internet of Things (IoT) are as follows.

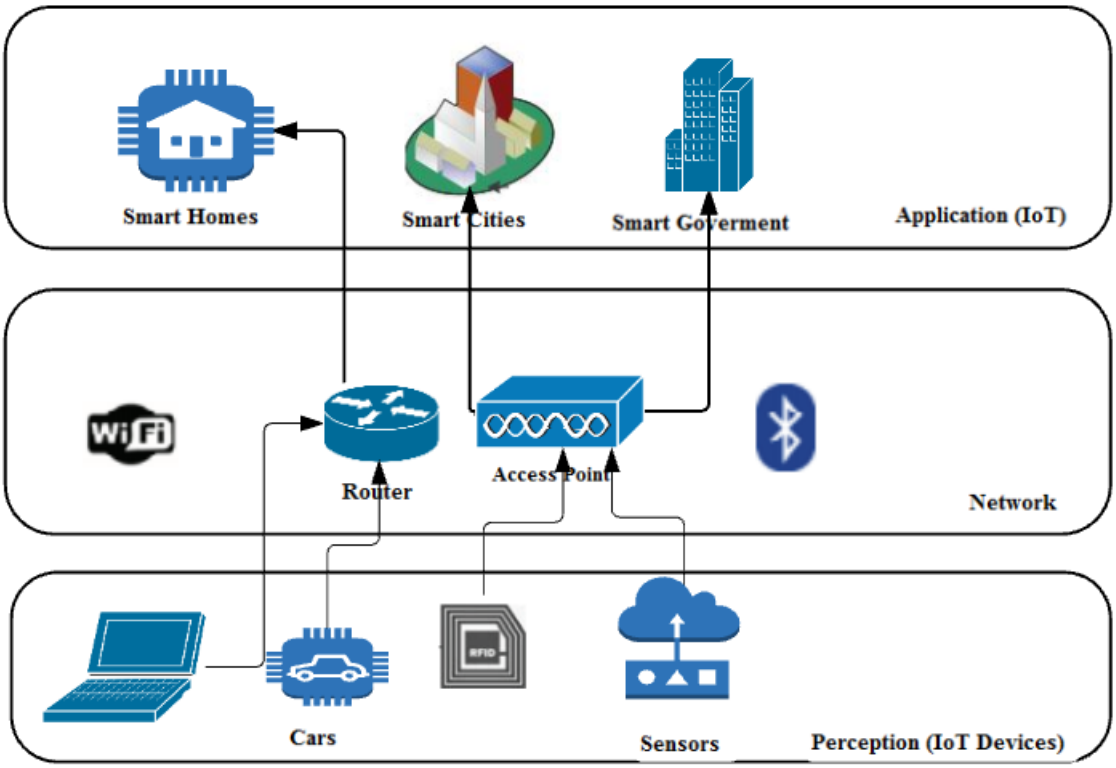

Figure 2.1: Traditional Internet of Things (IoT) Architecture

2.2.1 Insecure Web Interface

Insecure web interfaces are prevalent as the intent is to have these interfaces exposed only on internal networks, however threats from the internal users can be just as significant as threats from external users. An insecure web interface can be present when issues such as account enumeration, lack of account lockout or weak credentials are present. Attacker uses weak credentials, captures plain-text credentials or enumerates accounts to access the web interface used to configure an IoT device. Insecure web interfaces can results in data loss or corruption, lack of accountability, or denial of access and can lead to complete device take over.

2.2.2 Insufficient Authentication/Authorization

Many IoT devices are secured with low quality password such as 1234, qwerty etc. In addition, manufactures sets IoT devices with the weakest possible generic login credentials, and with no prompt for users to change the credentials. Attacker uses weak passwords, insecure password recovery mechanisms, poorly protected credentials or lack of granular access control to access a particular interface. Insufficient authentication/authorization can result in data loss or corruption, lack of accountability, or denial of access, and can lead to complete compromise of the device and user accounts.

2.2.3 Insecure Network Services

Some IoT devices are built on insecure kernel or use insecure services like Telnet, FTP etc. Attacker uses vulnerable network services to attack the device itself or bounce attacks off the device. Insecure network services can result in data loss or corruption, denial of service or facilitation of attacks on other devices.

2.2.4 Lack of Transport Encryption

Failure to use transport encryption to protect the data and credentials sent over the network. This allows data to be viewed as it travels over local networks or the internet. Lack of transport encryption is prevalent on local networks based on the assumption that local network traffic will not be widely visible, however in the case of a local wireless network, misconfiguration of that wireless network can make traffic visible to anyone within the range of that wireless network. Attacker uses the lack of transport encryption to view data being passed over the network. Lack of transportation encryption can result in data loss and depending on the data exposed, could lead to complete compromise of the device or user accounts.

2.2.5 Privacy Concerns

Some IoT devices sends recorded data to third parties, where users give their approval for collection and storage of data without having adequate information. Attackers uses

multiple vectors such as insufficient authentication, lack of transport encryption or insecure network services to view personal data which is not properly protected or is being collected unnecessarily. Collection of personal data along with a lack of protection of that data can lead to compromise of a user's personal data.

2.2.6 Insecure Cloud Interface

An insecure cloud interface is present when common weak/default credentials are used or account enumeration is possible. Attackers uses multiple vectors such as insufficient authentication, lack of transport encryption, and account enumeration to access data or controls via the cloud website. An insecure cloud interface could lead to compromise of user data and control over the device.

2.2.7 Insecure Mobile Interface

An insecure cloud interface is present when common weak/default credentials are used or account enumeration is possible. Attackers uses multiple vectors such as insufficient authentication, lack of transport encryption, and account enumeration to access data or controls via the mobile interface. An insecure mobile interface could lead to compromise of user data and control over the device.

2.2.8 Insufficient Security Configurability

Security features such as password policy enforcement, data encryption and access control are not strictly adhered to in IoT. In addition, an IoT user might not have the technical knowledge to securely configure an IoT network. Attacker uses the lack of granular permissions to access data or controls on the device. The attacker could also use the lack of encryption options and lack of password options to perform other attacks that could lead to compromise of the device or data (data loss).

2.2.9 Insecure Software/Firmware

Contents of software updates can be changed or replaced before they get to automatically-updating devices. This can allow an unauthorized user to run any code on the device including back-doors or the data collection malware. In addition, the lack of ability for a device to be updated presents a security weakness; devices should have the ability to be updated when vulnerabilities are discovered. Software/firmware updates can be insecure when the updated files themselves and the network connection they are delivered on are not protected. Software/Firmware can also be insecure if they contain hardcoded sensitive data such as credentials.

Attacker uses multiple vectors such as capturing update files via unencrypted connections, the update file itself is not encrypted or they are able to perform their own malicious update via DNS hijacking. Insecure software/firmware could lead to compromise of use data, control over the device, and attacks against other devices.

2.2.10 Poor Physical Security

Attackers uses vectors such as USB ports, SD cards or other storage means to access the operating system and potentially any data stored on the device. The attacker needs to have physical access to the device to carryout an exploit, hence addressing this vulnerability is out of scope of our research.

2.3 IoT Privacy Challenges

The services provided by IoT devices majorly makes use of data to provide the services that drives the smart experiment. Enormous amount of data is collected, stored and transferred over the network to enable basic functionalities of these devices.

In addition to the risks to security, privacy risk is highly associated with the Internet of Things. Some of these risks involve the direct collection of sensitive personal information, such as precise geolocation, financial account numbers, or health information - risks already presented in traditional Internet.

Major privacy concerns in IoT include facilitation of the collection of large amounts

of consumer data, using data in ways unexpected by the consumer, and security of data. A major difference between traditional Internet and the IoT is the amount of data being collected by the user [11]. Data universally collected in IoT can be used to build an invasive profile of an IoT consumer. Users agree to terms of service at some point, but never read through the privacy agreement document before giving their consent.

In addition, manufacturers or hackers could use a connected device to virtually invade a user's home. An example is intercepting unencrypted data from a smart meter device to determine what television show someone was watching at a specific moment [12]. This paper further discuss some prior security measures and techniques that have been developed to address some of these security issues.

Chapter 3

RELATED WORKS

Researchers have deployed several measures to address security issues in each layer of the IoT architecture. IoT requires security measures at all three layers; at physical layer for data gathering, network layer for routing and transmission, and at application layer to maintain confidentiality, authentication and integrity. Existing solutions to enhance security in IoT are divided into the following approaches;

3.1 SDN-Based and IoT Architecture Security Mechanisms

Jacob C. et al [13] used software defined networking (SDN) for early detection and elimination of ARP spoofing. They proposed a security framework that detects ARP spoofing attack early before the attack impacts other hosts in a network. This approach augments simple MAC-learning protocols on OpenFlow-enabled switches by hashing hosts' physical address with an appropriate IP:Port association to deny ARP spoofing at real-time.

Seugwon et. al proposed an application framework, FRESCO that exports scripting API that enables network administrators to code security monitoring and threat detection logic [14]. The research aimed to enable the modular design of complex openflow enabled network security services that can be built from smaller sharable libraries of security functions. The researchers implemented two components in an open-source OpenFlow controller, NOX; an application layer that provides an interpreter and APIs to support composable application, and a security enforcement kernel (SEK), that enforces the policy actions from developed security applications. This research is similar to our research in terms of implementing additional components in an OpenFlow controller to enhance network security. Our research integrates new components (vulnerability scanner, vulnerability mitigation and firewall) into existing components in the

POX controller.

Ahmed S. et al proposed an intrusion detection and prevention mechanism by implementing an intelligent security architecture using random neural networks (RNNs) [15]. The application's source code is instrumented at compile time in order to detect out-of-bound memory accesses, which is based on creating tags, to be coupled with each memory allocation and then placing additional tag checking instructions for each access made to memory. This helps to detect the presence of any suspicious sensor node within the system operating range and anomalous activity in the base station.

3.2 Authentication measures

Abhijit P. et al [16] proposed an hybrid lightweight and robust encryption design for security in IoT. The research points out how ciphers known to provide optimum security have gate counts that makes them unsuitable for application such as IoT devices, due to the constraints in IoT devices in terms of computational power and memory capacity. The research aimed to strengthen the Light Encryption Device (LED) ciphers by including SPECK key that produces perfectly 13 keys of 64 bits each meeting the primary requirement of LED cipher.

Xin M., proposed a hybrid encryption technique (cryptographic paradigm) that provides the benefit of the symmetric key and asymmetric key performance [17] for IoT security. The approach focused on securing the application layer of the IoT to ensure information integrity, confidentiality, non-repudiation on the data transmitted in IoT by using a mixed encryption algorithm; Advance Encryption Standard (AES) and Elliptic Curve Cryptography (ECC) algorithm. Messages and data sent and received over the IoT network are encrypted. The ECC algorithm was used as digital signatures and AES were used to encrypt the data.

Padraig F. proposed a protocol that combines zero-knowledge proofs and key exchange mechanisms to provide secure and authenticated communication in static machine-to-machine networks [18]. The protocol requires a prior knowledge about the network setup and structure, and it guarantees perfect forward secrecy. Zero-knowledge proofs (ZKP) are challenge/response authentication protocols, in which parties (each IoT net-

work) are required to provide the correctness of their secrets without revealing any information which could be used to help another party deduce these secrets.

Quangang W. [19] , implements the use of cipher security certificate. The certificate provides a method of one time one cipher between communicating parties (sensor nodes in IoT). It uses a lightweight encryption or decryption method using time stamp technology; timeliness in the two communication nodes is guaranteed.

Pedro M. et al demonstrated the feasibility of using Extensible Authentication Protocol (EAP)/ Protocol for Carrying Authentication for Network Access (PANA) [20] in devices with constrained capabilities and Internet-Protocol (IP)-based networking connectivity such as the IoT devices . The researchers designed a version of PANA to provide scientific community with the first light-weight inter-operable implementation of EAP/PANA for IoT devices in the Contiki operating system.

Sachin B. et al [21] proposed an embedded security framework as a feature of software/hardware co-designed methodology that help designers and developers to deliver more secure devices. The research adopts an hardware software based security architecture for IoT that aimed to serve as the best trade off cost/efficiency or security/performance using lightweight cryptography, physical security in terms of trusted platform module, standardized security protocols, secure operating systems, and secure storage.

3.3 Federated Architecture

To overcome the heterogeneity of various IoT devices, software and protocols, Anggorojati B. [22] proposed a model that depicts access delegation realized by means of a capability propagation mechanism, and the incorporation of contextual information as well as secure capability propagation under federated IoT environments. It uses identity-based capability-based access control approach in securing IoT. The model takes into consideration the flexibility and scalability that are key features in IoT systems.

Leo M., [23] defines the security needs by proposing a federated model to design. The models aims at securing the authenticity and integrity of the software installed on the IoT device by defining policies and standards to ensure security; and provides mechanisms to enforce such policies are followed. E.g. Software must be authorized to

run on the devices and has to be signed by an entity that authorized for it, creation of policies that can limit privileges of device, components and applications so that they only have access to the needed resources.

Michael W. et al [24] proposed the use of smart edge IoT devices for safer, rapid response with industry IoT control application in which the control systems are remotely managed. The control system leverages devices connected to the Internet using a gateway device that can issue suitable control sequences and sends notifications of unusual usage critical events.

3.4 Trust Mechanism

Since IoT devices can physically move from one owner to another in a network, trust can be established between both owners to enable smooth transition of the IoT devices with respect to access control and permissions.

Mahmoud R. [2] presents the concept of mutual trust for inter-system security in IoT by creating an item-level access control framework. Trust is established from the creation to operation, and transmission phase of the IoT. The trust is established by the creation key and the token mechanism. A new IoT device added to the network is assigned a creation key by an entitlement system, and the token is created by the system administrator (the token is combined with the unique identifier of the device). This mechanism ensures the change of permissions by the device itself, if it is assigned a new owner.

3.5 IoT Security Services

To address some of the security challenges facing the Internet of Things, companies and organization have developed tools that enables users to know if devices in their home network or networks in their perimeter are vulnerable. Beyond Trust [25] implemented a scanner, Retina IoT (RIoT), a free vulnerability scanner that identifies Internet of Things (IoT) devices, and their associated vulnerabilities. They claim the tool identify high-risk IoT devices, check for default or hard-coded passwords and generate clear IoT vulnerabilities reports and remediation guidance. Users are only able to scan their

devices after they already have communication access in the network, which gives an attacker an opportunity to exploit vulnerable IoT devices.

Bullgard [26], a UK-based consumer security company developed a free IoT scanner that allows IoT users to scan a home network for smart devices that are exposed to the internet and could be vulnerable. The scanner scans the user's public IP address to detect if it is reachable from the internet using data from Shodan [27]. It is reported that the scanner can be used to check security cameras, baby monitors, Smart TVs and wearables that may be visible to hackers, flags accessible devices along with details of potential vulnerabilities.

Linda et al. [28] presented three different types of scans for vulnerable devices in the Internet of Things. The first uses Shodan to find vulnerable device such as routers; the second uses Masscan to find devices vulnerable to Heartbleed (the heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software) vulnerability, and the third uses Nmap and PFT to find and connect to vulnerable networked printers.

An approach similar to this research's approach has been used in detection of malware in USB devices. Shue C. [29] forces a USB drive that is plugged in to undergo a series of virus scans prior to it being made available to the user or any other part of the system.

Our system obviously differs in that we are concerned with network access rather than auto-running malware, but also in that we are also focused on healing the vulnerabilities and providing useful information to the user. Other approaches include the use of protocol simulators [30] where there is huge variety of device end-points and interfaces to validate. Data recorders [31] used for smart validation across device sets whereby the recorded data can be played across different device end points automatically, which in turn can be a great enabler in compatibility testing of apps across different device sets and communication layers.

In the works in [14], [15], [16], [17], [18], the security provided requires that the manufacturer, architect, or administrator performs time-intense and highly technical tasks. In many cases, the person responsible will not perform these due to factors

such as cost or inadequate technical expertise knowledge. Although, the tools [25] [26] developed by companies enable users scan for vulnerabilities in devices in their private network, the tool provides an after-installation security service for consumers, which gives an attacker an opportunity (the time between installation and the time the consumers decides to scan for vulnerable devices) to exploit vulnerable devices in a user's network.

Our approach provides a during-installation security service, such that IoT devices are scanned for vulnerabilities immediately they are leased an Internet Protocol (IP) address to communicate; the devices are blocked from communicating until the vulnerability scan is complete. It also runs scans on devices with static IP addresses in which the system checks if a source and destination address of a packet has been scanned for vulnerabilities. This approach does not give an open window for attackers to exploit a vulnerable device in the network. In addition, our approach does not rely on the expertise and diligence of users or manufacturers, and provides confidence to the user that their system is unlikely to be compromised easily.

Chapter 4

SYSTEM DESIGN

4.1 Security Model

A vulnerable IoT device in a network creates an attack surface by which an attacker can exploit vulnerabilities. The attacker can then launch an attack such as denial-of-service attack on the entire IoT network, and even shut down services on an outside domain as discussed in section 1.1. SeeSec is intended to operate in applications of the Internet of Things. Examples of these applications are smart homes, wearables, and smart cities.

We introduce a Software define networking (SDN)-based architectural framework to enhance security in the Internet of Things. SDN is an approach to using open protocols to apply globally aware software control at the edges of the network to access network switches and routers that typically would use closed and proprietary firmware. Our approach uses a centralized SDN controller that controls and manages the entire IoT network. To eradicate potential attack surface in an IoT device, the security service provided by SeeSec is installed in an SDN controller. The SDN controller defines and communicates policies to specify traffic behaviour on OpenFlow switches.

The SDN controller is connected to an OpenFlow-enabled switch. An openflow-enabled switch is an hardware device that forwards packets in a SDN environment. The openflow switch serves as the data plane (implements how traffic is forwarded), which communicates with the SDN controller (the control plane that decides how traffic must be forwarded) through a southbound API (Application Programming Interface).

Our threat model includes vulnerability scans in both existing IoT device in a network, and a newly added IoT device. The IoT devices that are being installed are assumed to be potentially vulnerable but not actively malicious. In other words, the IoT device may be running software with a known vulnerability. The security services provided by the SeeSec takes place on two levels;

- **Connectivity Level** - prevents vulnerable devices from connecting to the IoT network. This is especially a feature of devices with non-static IP addresses

- **Service Level** - prevents vulnerable devices from communicating with other devices in the network. For devices already in the network, the framework checks if they have undergone a vulnerability scan before it allows the devices to carry out any form of communication (mainly for devices with static IP address).

The focus of this study are on wireless device connections, but it will also apply to wired device connections. SeeSec does not address physical threats associated with IoT devices. It is not designed to detect devices that have been compromised prior to being installed or that are being installed by a malicious user. It aims to enable non-technical (and technical) users to configure secured IoT network by preventing vulnerable devices in the network.

The framework presents a shift in the traditional IoT architecture, by separating the control plane (network logic) and the data plane. With this, it aims to both decrease the numbers of vulnerable devices active in an IoT network and to provide a baseline security for the Internet of things. SeeSec does not aim to redesign existing cryptographic algorithms, create new software patches, nor detect anomalies in network traffic. The next section explains the process flow of the proposed security framework, SeeSec.

4.2 Process Flow

Figure 4.2 shows the process flow of the proposed security framework. For an IoT device to join the network, the device sends a request to the DHCP (Dynamic Host Configuration Protocol) server to be assigned an IP address. The DCHP server protocol automatically provides an Internet Protocol (IP) host with its IP address and other related configuration information such as the subnet mask and default gateway [32]. The DHCP server leases an IP address to the device and initiates a vulnerability scan. This is done by invoking the scan server which in turn connects to the integrated penetration testing tools and custom vulnerability scanners via a REST API, RMI and RPC appropriately.

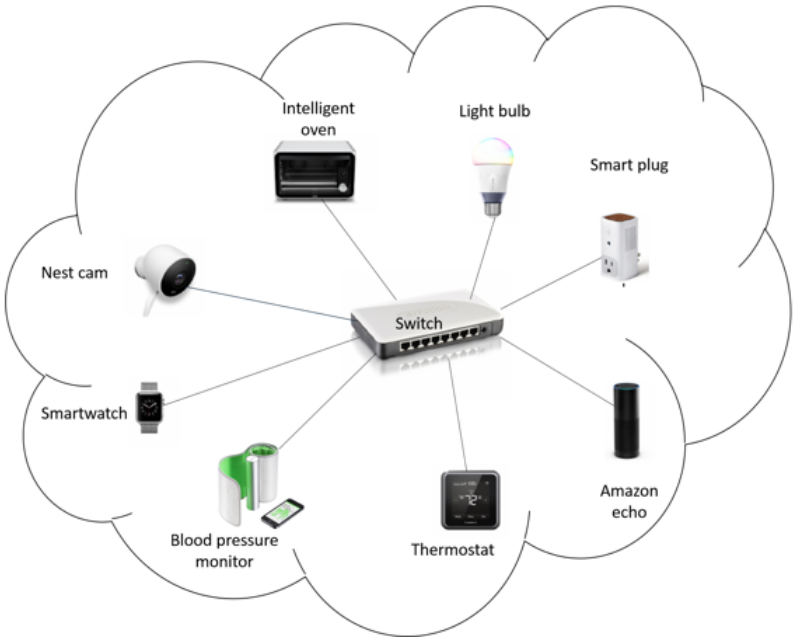

Figure 4.1: Interconnected IoT Devices

Implementation will include integration of both custom scanning tools and professional penetration testing tools with the security framework. The custom vulnerability scanners will aim to detect IoT specific vulnerabilities. This will enable an optimal performance of the system in terms of scan time and eliminating false positives and false negatives. The scanning tools will be executed simultaneously to achieve a near real time scan, and avoid delays to join an IoT device to the network.

The scan server launches the vulnerability scans and analyze the scan results from the scanning tools to determine if an IoT device is vulnerable. IoT devices that are perceived to be vulnerable are blacklisted and non-vulnerable devices are whitelisted. The scan server updates an Access Control List (ACL) managed by the DHCP server and host tracker component of the system. The ACL is utilized by the firewall component of the framework to define rules and update the flow tables of the OpenFlow switch. The proposed framework also works in the case of static IP addresses as those devices will not be whitelisted yet, so all traffic to or from that IP address will be blocked until

the device is whitelisted. On the first attempt of an IoT device with a static IP address to communicate in a network, the device undergo all required scans; this is invoked by the host tracker component.

After the scan is complete, the system examines the vulnerabilities and then determines if it knows any methods for fixing those vulnerabilities. If it does, it will apply the fix and if the fix is successful for all the vulnerabilities on a device, the device will be whitelisted and granted access to the network. When a device is deemed vulnerable, an email will be sent to the user explaining the vulnerabilities and offering suggestions to the user for fixing the vulnerabilities. This paper further discusses the process flow of the core component (scan server) and other relevant design decisions in sections 4.3 and 5.2.

4.3 Design Goals

The overall design goal of the proposed security system is to provide a usable framework through which non-technical users can securely set up an IoT network regardless of the underlying security mechanisms. In this light, the framework should have the following functionality:

4.3.1 User Friendly

SeeSec should resolve detected vulnerabilities that can be automatically resolved and provide the user with suggestions on how to fix the unresolved vulnerabilities. While the primary focus is to enable non-technical users to securely set up the network, the vulnerability resolution suggestions provided by the SeeSec must be presented in a way user's can easily understand. In addition, it is important that the framework is easy to learn, quick to use, simple to remember and easy to navigate [33]. Providing a remote security service to users, whereby users' openflow-enabled switches or routers are able to automatically connect to remote controllers is one way to achieve this goal.

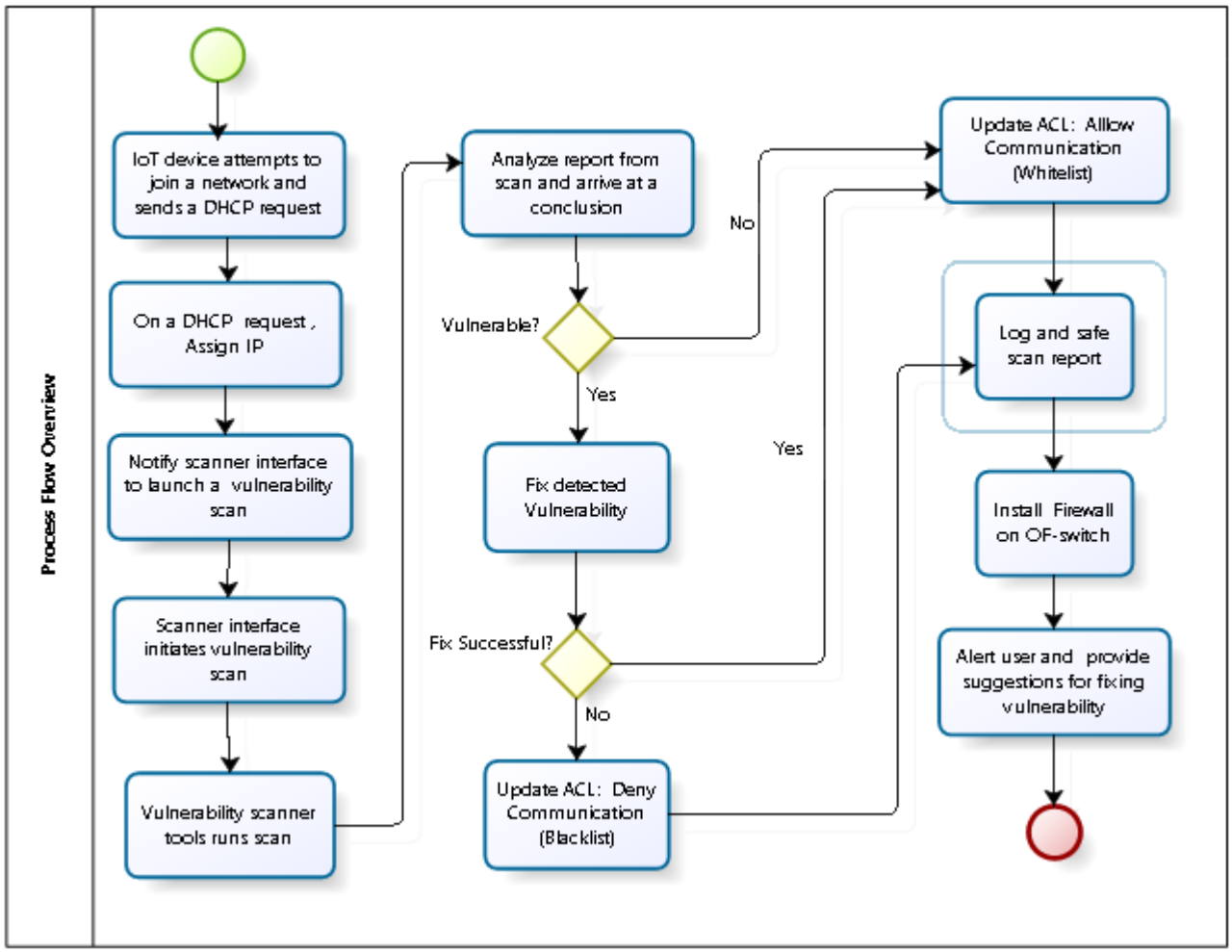

Figure 4.2: Process Flow of Proposed Framework

4.3.2 Scalability

SeeSec design should be easily enhanced with respect to continuous improvement. It detects vulnerabilities in IoT devices using integrated vulnerability scanners. SeeSec should have an interface where new implemented vulnerability scanners can be easily integrated with the existing system. Adding new modules that carry out new functionalities in the system should be easy. In other words, a change in the implementation of one of the components or adding a new component to the system should not create a need to make major changes in other components of the system. In addition, SeeSec should be able to support concurrency in terms of adding a large number of IoT devices to a network simultaneously (the system should be able to run vulnerability scans on multiple devices requesting to join a network at once).

4.3.3 Access Control

SeeSec should ensure all communication channels are blocked to and from vulnerable IoT devices. The approach in implementing a firewall or communication flow control must be very effective. In addition, it must be ensured that all devices in the IoT network has been scanned for vulnerabilities.

4.3.4 Availability

SeeSec should be deplorable in a way it is readily available to its users. The framework should enable the establishment of a centralized SDN controller which could be configured remotely (managed by a service provider) or locally (managed by the network administrator).

4.4 Assumptions

It is assumed that the SDN controller used to manage the IoT network is secured and not susceptible to attackers. We also assume user's routers or switches are openflow-enabled.

4.5 Proposed IoT Network Architecture

Our system proposed an SDN-based architecture for IoT to enhance security (Figure 4.3). The core components of the proposed architecture for the purpose of this research include: SDN controller, OpenFlow-enabled switch, Scan server and Vulnerability scanning tools or program. The function of some of the components of the POX controller used for this research is discussed in subsection 4.5.4.

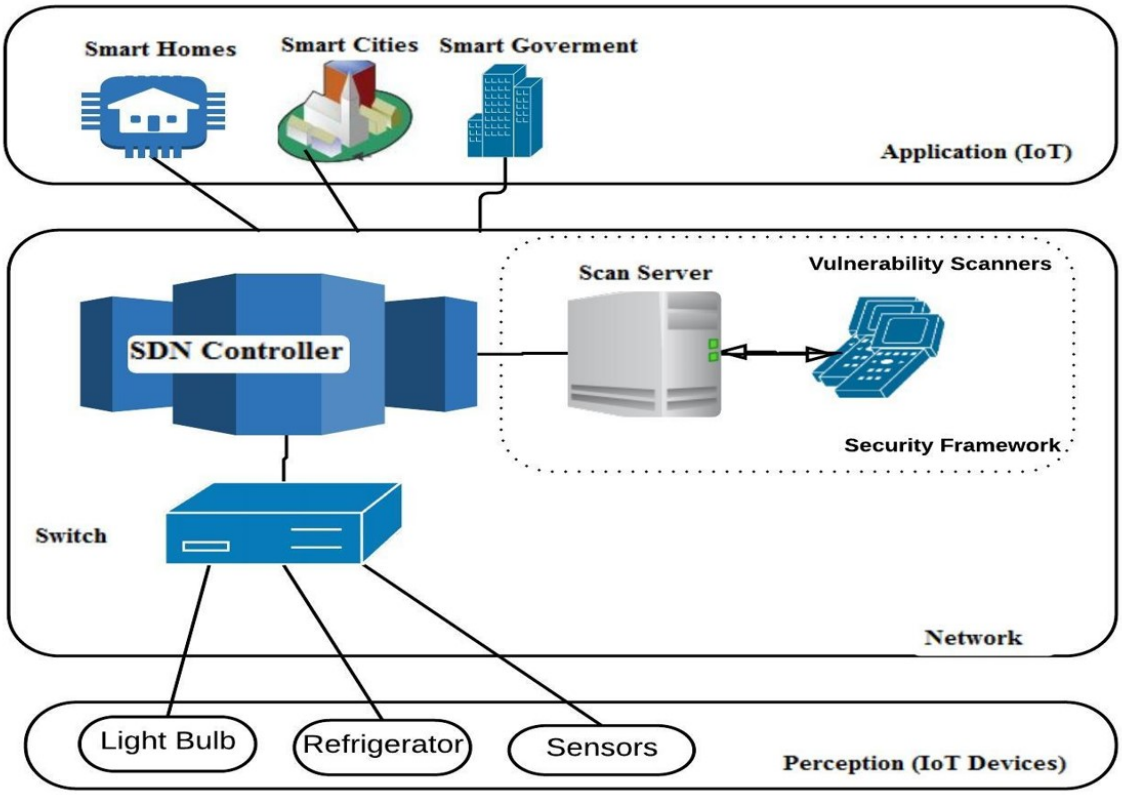

Figure 4.3: Proposed Framework for IoT Security

4.5.1 Software Defined Networking (SDN)

Software Defined Networking (SDN) is an approach to using open protocols to apply globally aware software control at the edges of the network to access network switches and routers that typically would use closed and proprietary firmware. SDN emerged a a strategy to increase the functionality of the network, reducing costs, reducing hardware complexity, and enabling innovative research [5].

SDN as an emerging paradigm in networking divides the network architecture into three distinct layers: data [34] or infrastructure [5] layer consists of network devices (e.g., switches, routers, virtual switches, wireless access point), a control layer consists of SDN controller(s) (e.g., NOX, POX, Floodlight, Beacon, Open daylight, etc.) and an application layer that includes the applications for configuring the SDN (e.g., Access control, packet analysis, traffic/security monitoring, energy-efficient networking, and management of the network).

SDN technology pulls a network's control plane into a dedicated SDN controller which manages and mediates all functions and services on virtual and physical networks. SDN is able to provide unprecedented programmability, automation, and network control by decoupling the control and data layers, and logically centralizing network intelligence and state [35].

The SDN architecture also has the ability to extend the security perimeter to the network access end point devices (access switches, wireless access points etc.), by setting up security policy rules to network devices via the OpenFlow protocol [36]; the SDN controller builds a global view by establishing connection with the OpenFlow switches. Network applications in the application layer of the SDN architecture communicate with the SDN controller via an open interface and define network-wide policies based on a global view of the network provided by the controller.

SDN enables centralized network control and traffic management, which will result in automated network security framework that is more adaptive and scalable. Controllers and openflow switches are able to identify packet attributes which allows for automated blocking or offloading of traffic in Denial of Service (DoS) attacks. Figure 4.4 shows

redefined IoT architecture using the SDN paradigm. The SDN controller, openflow-enabled switches and scan server acts as its core component.

4.5.2 OpenFlow

OpenFlow is a standard communications interface defined between the control and forwarding layers of an SDN architecture. OpenFlow allows direct access to and manipulate of the forwarding plane of network devices such as switches and routers, both physical and virtual.

OpenFlow enables networks to evolve, by giving a remote controller the power to modify the behavior of network devices, through a well-defined forwarding instruction set. For the purpose of this research, we used HP2920 24G POE switch, an openflow-enabled switch as the network device in the SDN architecture. An OpenFlow switch is a software program or hardware device that forwards packets in a SDN environment. The OpenFlow-enabled switch encapsulates and forwards the first packet of a flow to an SDN controller, enabling the controller decide whether the flow should be added to the switch flow table; forwards incoming packets out the appropriate port based on the flow table; and drops packets on a particular flow, temporarily or permanently, as dictated by the controller [37].

4.5.3 SDN-based IoT Architecture

As shown in Figure 4.4, the proposed architecture is a layered architecture consisting of three basic layers; application/services layer, controller layer (control plane), and data plane layer called forwarding layer which consists of forwarding devices. These SDN layers communicate with each other via open APIs called Northbound Interface (NI) API and Southbouth Interface (SI) API [38].

To identify the different elements of our proposed architecture as clearly as possible, we now present the essential terminology used in the architecture.

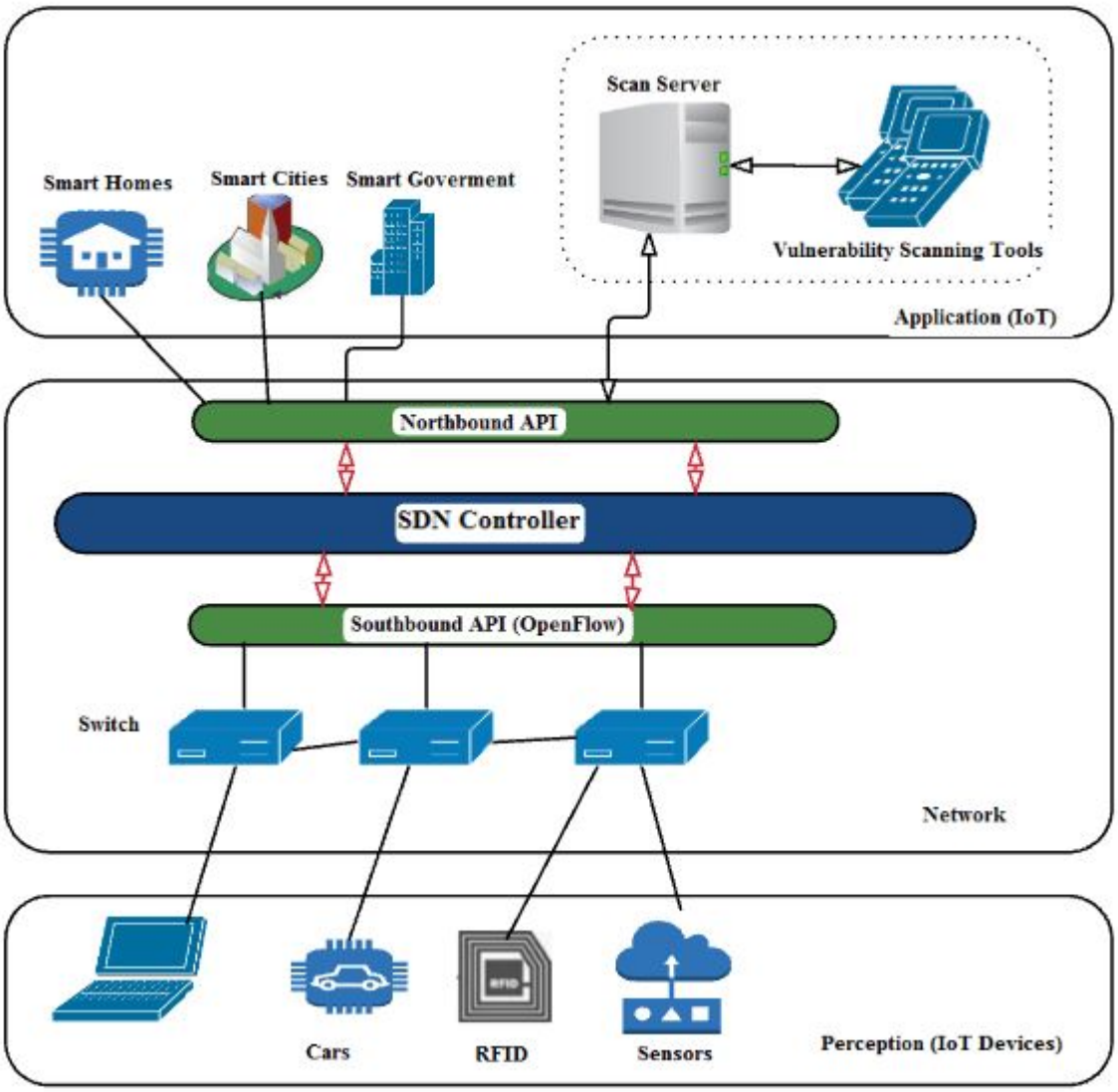

Figure 4.4: Redefined IoT Architecture, based on the SDN Paradigm

4.5.3.1 Application Layer (AP)

The application plane also called management plane consist of applications that leverage the functions offered by the NI to implement network control, and operation logic. Essentially, a management application defines the policies, which are translated to southbound-specific instructions that program the behavior of the forwarding devices.

4.5.3.2 Northbound Interface (NI)

The interaction between application and control plane is provided through Northbound Interface. NI APIs abstracts the low-level instruction sets, and implementation of forwarding devices.

4.5.3.3 Control Plane (CP)

Control plane is the decoupled entity from the distributed forwarding devices and logically centralized on a server. CP programs the forwarding devices through southbound interfaces. CP defines rules/instruction set for forwarding devices, hence the POX SDN controller is the "network brain" (Figure 4.3), and all control logic rests in the applications and controllers, which form the control plane.

4.5.3.4 Southbound Interface (SI)

Southbound interfaces provide a communication protocol between CP and forwarding device through the SI instruction set. Well established SI protocol help controller in programming forwarding devices and formalize rules for interaction between the two planes (CP and DP).

4.5.3.5 Forwarding Devices (FD)

Network core devices, either software based or hardware based performs fundamental network operations. The forwarding devices act on the basis of rules provided by the controller on the incoming packets or flows. These rules are defined by southbound

interfaces (OpenFlow), and are installed in the forwarding devices by the POX SDN controller implementing the southbound protocols.

4.5.3.6 Data Plane (DP)

Forwarding devices (routers, switches, gateways etc.) are interconnected through a physical medium such as wired cables or wireless radio channels, which defines a phycical interconnection within a network.

For the purpose of this research, we used POX SDN controller to define the data flows that occur in the OpenFlow-enabled switch (the data plane).

4.5.4 POX SDN Controller

POX is an open source development platform for Python-based SDN control applications, geared towards research and education. POX enables rapid development and prototyping, and can be useful for writing networking software. POX currently communicates with OpenFlow 1.0 switches and includes special support for the Open vSwitch extensions (used in the mininet simulation). The openflow-enabled switch (HP2920) used for this research supports OpenFlow versions 1.0 and 1.3.

POX requires Python 2.7, and supports Windows, Mac OS, and Linux operating systems. POX comes with a number of stock components, some of which provides core functionality, and provide convenient features. For the purpose of this research, the POX components used includes: py, openflow.discovery, proto.dhcpd, topology.dhclient, host_tracker, forwarding.l2_learning, and misc.firewall. A python script (start.py) that launch these components was implemented, this enable us to launch only one component that trigger other required components.

4.5.4.1 Py

This component causes POX to start an interactive Python interpreter that can be useful for debugging, and interactive experimentation.

4.5.4.2 Forwarding.l2_learning

This component makes OpenFlow switches act as a type of L2 learning switch, which learns L2 addresses; the flows it installs are exact matches on as many fields as possible. This component was used in our research for the purpose of creating a virtual IoT network using Mininet simulation tool.

4.5.4.3 Openflow.discovery

This component sends specially-crafted LLDP (Link Layer Discovery Protocol) messages out of OpenFlow switches so that it can discover the network topology. It raises event (which can be listened to by hardware openflow switches) when links go up and down. This component enable connection between the POX controller and the HP2920 openflow-enabled switch.

4.5.4.4 Proto.dhcpd

This component serve as the DHCP server of the IoT network. By default, it claims to be 192.168.0.254, and was configured to serve clients (IoT devices) address in the range 192.168.0.13 to 192.168.0.253, claiming itself to be the gateway and the DNS server. For the purpose of our research, we configured this component to initiate a vulnerability scanner on IoT devices after responding to a DHCP request made from an IoT device (i.e. after leasing an IP address to an IoT device in the network).

Right before leasing issuing an address, the DHCP server raises a DHCPLease event which we csn listen to if we want to learn or deny address allocations. This feature makes it easy to decline requests from specific IoT devices (identified by their MAC address) known to be vulnerable; assuming there is no malicious user spoofing the MAC address of an IoT device.

4.5.4.5 Host_tracker

This component attempts to keep track of hosts in the network, where they are and how they are configured (their MAC/IP addresses). When things change, the compo-

nent raises a HostEvent. Host_tracker functions by examining packet-in messages, and learning MAC and IP bindings that way. We periodically ARP-ping hosts (devices) in the network to see if they're are still there, and relies on packets coming to the controller. In addition, this component is configured to check if a host (IoT device) in the network has been scanned for vulnerabilities. For every packets sent in the network, the host_tracker component checks if the packet source and destination MAC address have been scanned. If the source or destination host have not been scanned, this component launches a vulnerability scan on the device by retrieving the IP address of the device using it's MAC address.

Chapter 5

IMPLEMENTATION

This research presents a shift in the traditional IoT architecture by separating the control plane from the data plane; the Software Defined Networking (SDN) paradigm. This research aim to enhance security in IoT by preventing vulnerable devices from joining the network to avoid providing attackers the opportunity to launch an attack.

The main function of the framework is to scan for and fix vulnerabilities in IoT devices as shown in figure 4.2. Note that while the current system design uses SDN for its ease of deployment and testing, it would be possible to create a similar system using traditional routers. The security service provided to IoT in the implemented system is driven from the SDN controller, POX.

This chapter discusses how the SDN controller is used to enhance security of IoT by implementing a security framework, a scan server that scans for vulnerabilities in an IoT device, and a mitigation module that resolves detected vulnerabilities when possible (when the server has the resources to resolve a specific IoT vulnerability such as insufficient authentication resolution, which can be done by changing the password of an IoT device).

Figure 5.1 shows an overview of the implemented components and their connections. The broken, one-directional arrow depicts the component triggering an event in the receiving component, while the solid arrows depicts a bidirectional communication between the two components.

5.1 SDN Controller

The SDN controller can be centralized or distributed (we use a centralized SDN controller for the purpose of this research). It exchanges protocol updates and maintains the routing table through exchanging updates between other controllers. The central-

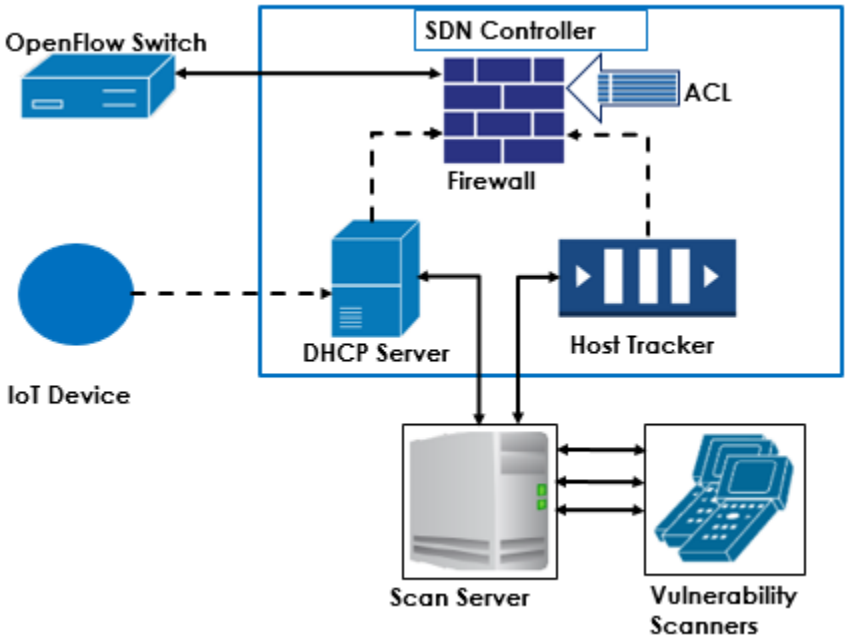

Figure 5.1: Diagram showing the Implemented Components of the Proposed System Architectural Design

ized controller has network intelligence and global view of the network, and can manage the entire network. The network devices has to only accept instruction from the controller. The instructions are channeled through an OpenFlow switch. The controller consist of components that are used to control and manage the network to enhance security. We configure the DHCP server sub-component of the SDN controller to initiate vulnerability scan and implement a firewall component to control the communication of hosts.

We reconfigured two components (DHCP server and Host-tracker) of the POX controller to enhance IoT security. In addition, we implemented a new component, a firewall that disable communication to and from vulnerable IoT devices.

5.1.1 DHCP Server

The DHCP subcomponent handles DHCP requests from IoT devices. The DHCP server leases an IP address to the device and initiates a vulnerability scan via an interface provided by a scan server.

5.1.2 Host Tracker

This is a sub-component of the POX controller. It tracks hosts in the network and detects host that has not been previously scanned in the network. When a host with a static IP address attempts to send a packet to another host in the network, this component is configured to check if the packet's source and destination address has been previously scanned for vulnerabilities. A vulnerability scan is launched via the scan interface for hosts that have not scanned for vulnerabilities. Hosts perceived as vulnerable are subsequently blacklisted, and non-vulnerable devices are whitelisted.

5.1.3 Firewall

The firewall is a subcomponent of the SDN controller that enforce security policies, and restricts unauthorized network access in the IoT network by filtering network traffic based on the predefined rules in the access control list. The firewall prevents all forms of communication to vulnerable IoT devices in a network. This component insert rules in the controller; and the controller subsequently updates the flow table of the OpenFlow switch. The flow table entry contains a set of rules (e.g. IP source) to match and an action list (e.g. forward to port, drop etc.) to be executed in case of a match.

5.1.4 Access Control List

The access control list is used to define rules, and it is utilized by the firewall. It helps the firewall to identify vulnerable devices in the network, providing a pair of source and destination MAC address that identifies where to restrict communication. Blacklisted devices are restricted from communicating with other hosts on the network. That is,

any packets sent by blacklisted host/device are dropped and attempts to send network packets to them are rendered void.

5.2 Scan Server

The scan server is the core component of implemented security framework, SeeSec. SeeSec incorporates a modular and composable design architecture. The scan server is the main component that provides security in the IoT network.

5.2.1 Software Development Life Cycle

SeeSec was implemented following several iterative and incremental development approach, the Agile methodology. The development process incorporate iterations of continuous planning, continuous testing, and continuous integration. This approach was used for development because it gives the ability to create and respond to changes in terms of system requirements and components required to implement an efficient framework.

The functional requirement of SeeSec kept evolving throughout the research process; the goals and components. The agile approach allowed easy addition of system requirements, in terms implementing vulnerability scanners that detects IoT-specific vulnerability. Each vulnerability scanner currently integrated with the system was implemented iteratively, and changes required to be made to existing components were addressed sequentially.

5.2.2 Scan Server Design

SeeSec is a security prog in the application layer of the architecture, provided in the SDN controller (control plane) of the network. This component provides an interface for the DHCP server component to send requests to launch vulnerability scans on an IoT device/host in the network, after leasing an IP address. The provided interface is also used by the host tracker component to launch vulnerability scans on IoT devices with static IP address. Figure 5.2 shows the overview of the processes run by the scan

server via the interface provided.

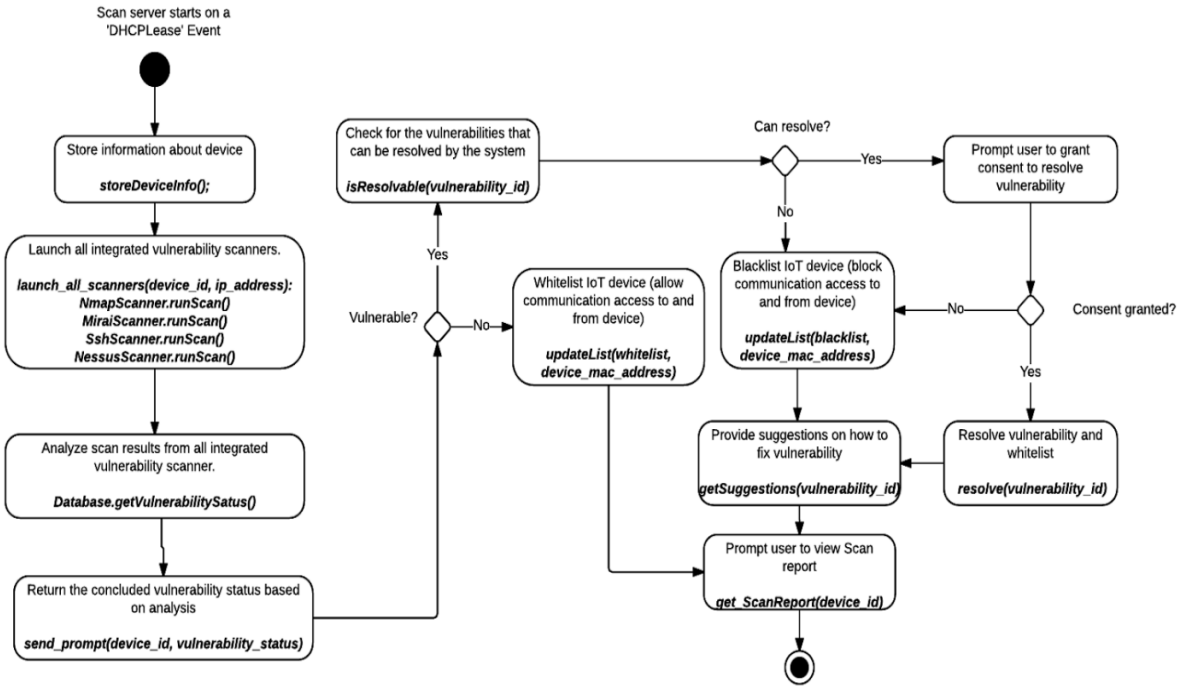

Figure 5.2: Scan Server Communication Diagram

The scan server connects to the integrated vulnerability scanning tool (Nessus) and custom vulnerability scanners (SSHScanner, Mirai, and NmapScanner) via REST API, and RPC respectively, to launch vulnerability scans. The integrated vulnerability scanners run sequentially.

The scan result from each integrated scanners is stored in the database, and analyzed to reach a conclusion about the vulnerability status of an IoT device. Each integrated vulnerability scanner stores scan report in the database via remote method invocation (RMI) using an interface implemented in the data access layer of the scan server architecture.

The access control list is updated accordingly based on the vulnerability status (vulnerable or non-vulnerable) of the scanned IoT device. Non-vulnerable devices are whitelisted, i.e. granted communication access to the network. For devices with vul-

nerabilities, the system checks if it has the ability to resolve the vulnerability (e.g. password change), and prompts users to give the system permission to resolve the vulnerability. SeeSec resolves the vulnerability if the user gives SeeSec consent to resolve a vulnerability, and suggestions on how to fix unresolved vulnerabilities is provided to the user. SeeSec blacklist an IoT device if one or more vulnerability exist in the device. For example, if a user decline to resolve a vulnerability on a device, the system provides remediation suggestions, and block communication access on that device. The system could also be configured to allow communication access in a vulnerable device if the user opt in to add an IoT device regardless of its vulnerability, however this will cause a drawback to the system's goal to enhance IoT security.

The system provides suggestions on how to resolve vulnerabilities the system cannot automatically fix on behalf of the user, and the user is alerted about fixes and unresolved detected vulnerabilities. Scan reports and other data (services running on a device, encryption algorithms used, common vulnerabilities found, etc.) stored in the database can be used for audit, and also analyzed to ease the process of further implementation (continuous improvement) of SeeSec.

5.2.3 Resolution Module

This subcomponent of the scan server is used to resolve vulnerabilities. It handles resolution of vulnerabilities that can be automatically resolved on behalf of the user. Current implementation of our system has the ability to resolve default password vulnerability for devices that can be remotely logged into. Additional vulnerability resolution module(s) in the system will be implemented in this component. In addition, this component also generates suggestions on how to fix unresolved vulnerabilities; the information is retrieved from the database based on the ID of the vulnerability existing in an IoT device.

5.2.4 Access Control list (ACL)

This subcomponent stores information about whitelisted and blacklisted IoT devices. The MAC address of blacklisted IoT devices and whitelisted devices are stored. The ACL is utilized by the firewall to install flow rules on the Openflow switch. The pseudo code below shows the implementation logic of this subcomponent.

```
for vulnerableDevice in blackList:
    for device in globalList: //Blacklisted and Whitelisted IoT device
        firewall.writerow([count, str(vulnerableDevice).strip(),
            str(device).strip()])
```

The firewall policy installed on the Openflow switch also disable communication between blacklisted devices, i.e. vulnerable devices cannot communicate with each other.

5.3 Database

One of the core component of SeeSec is the data storage (Database). The database stores the entire data of the system which include; IoT device BIOS, operating systems and version, encryption algorithms used in IoT devices, IoT specific vulnerabilities, vulnerability remediation suggestions, scan reports etc. Multiple components in the scan server connects to the database for read and write operations.

SeeSec is python-based (Python was used for implementation), and python provides database interfaces for a wide range of database engines. Some of the commonly used database engines include MySQL, SQLite, PostgreSQL, Oracle, Ingres, SAP DB, Informix etc. For the purpose of this research, we considered using PostgreSQL and SQLite database engine.

5.3.1 PostgreSQL

PostgreSQL is an open source relational database management system (RDBMS) used to store data securely, and return the data in response to requests from software ap-

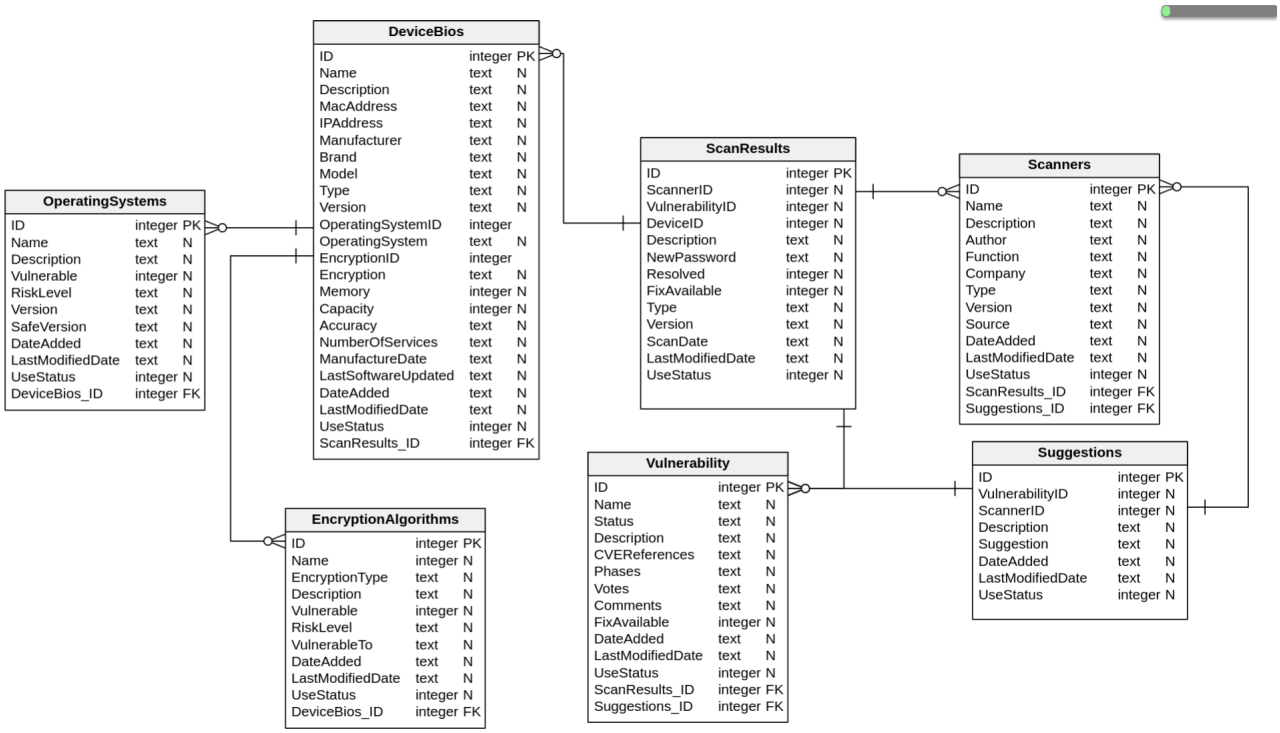

Figure 5.3: SeeSec Database Design Diagram

plications. It can handle workloads ranging from small single-machine applications to large Internet-facing applications or for data warehousing with many concurrent users. It supports functions and stored procedures, and it is ACID(Atomicity, Consistency, Isolation, Durability) compliant; ACID is a set of properties of database transactions.

Although PostgreSQL is a suitable to use as an internal storage for the implemented security framework, it will require high memory and CPU capacity to run. PostgreSQL is the recommended relational database for working with Python web applications, which could make it a good fit for SeeSec's database engine if user's will interact with the system via a web interface.

In addition, one of the design goals of SeeSec is to enable a remotely and locally deployable security service. PostgreSQL will be the best fit for database management for remote security service (e.g. when SeeSec is run as a third party system that provides security service remotely to consumers). Using PostgreSQL as internal storage when

SeeSec is deployed locally (in the user's home router or a network device) is not the best approach because of its high utilization of computing resources.

5.3.2 SQLite

SQLite is a server-less, zero-configuration, self-contained SQL database engine suitable for developing embedded applications. SQLite is open source which makes it freely available for use for the purpose of this research. Unlike many other SQL databases, SQLite does not require running a separate server process. Instead, SQLite stores all the data directly onto flat files that get stored on a computer disk. These files are easily portable across different platforms, making it a very popular choice for smaller and simpler database implementation requirements. Python comes with built-in standard library for sqlite3 support.

The SQLite database does not support high concurrency i.e. SQLite supports an unlimited number of simultaneous readers, but it will only allow one write at any instant in time. This is not a concern for the current implementation of the proposed system, because the vulnerability scanners are run sequentially and not simultaneously. SQLite database was suitable for the purpose of this research because it is light weight and it functions well as an in-memory data storage for embedded systems.

To enable easy continuous improvement of SeeSec in terms of database management, we used a python object relational mapping (ORM) library, Peewee, to manage data retrieval and storage. Peewee is an object-relational mapper, written in python, which can enable easy migration of the SeeSec's data storage to a more advanced database engine such as PostgreSQL, SQL server and MySQL. Peewee has a built-in support for PostgreSQL, SQlite and MySQL.

For the purpose of database security, in terms of protecting the system from attacks such as SQL injection attacks, we do not assemble the system's query using Python string operations. Instead we use the DB-API parameter substitution by putting the question mark sign, '?', as a place holder wherever a value is in use, and then a tuple of values is subsequently provided as the second argument to the cursor's execute.

5.3.2.1 Database Schema and Data

There are entities in the database that stores primary data, this include encryption algorithms used in IoT devices, operating systems, known IoT-specific vulnerabilities, and remediation suggestions for known vulnerabilities. Figure 5.3 shows the diagram of SeeSec database design.

The term 'primary' is used to denote the data is only updated when new events and findings occur in the Internet of things. For example, when a new IoT vulnerability is found, new vulnerability findings in an encryption algorithm or operating system version, implementation of additional vulnerability scanners, and enhancing quality of remediation suggestions provided to users. In addition, there are entities in the database that stores data specific to a scanned IoT device. The entities include scan results that stores reports from all vulnerability scanners, bio-data information of scanned IoT device such as the device's operating system and encryption algorithm. Other tables in the database include entities that stores the detailed information (report) of each integrated vulnerability scanner, ports used in an IoT device etc. We discuss (Figure 5.3) only the core entities of the database.

- **Encryption Algorithms**

  The encryption algorithms used in IoT devices are stored in this data table. The description and the vulnerability status of the encryption algorithms are stored. Information on encryption algorithms used in IoT devices and their vulnerability status was gathered from multiple sources [27][39], and a report on known vulnerable encryption algorithms were used to identify which algorithm can pose a risk to an IoT network if used in an IoT device.

- **Operating Systems**

  This stores information about operating systems used in IoT devices. Information such as the versions of the operating system, resources and their known vulnerability status (vulnerable or non-vulnerable). Data was gathered from [27][40][41].

- **Vulnerability**

  This stores information about known IoT-specific vulnerabilities. This information was sourced from the common vulnerability exposure (CVE) database [42]. We identified IoT specific vulnerabilities by searching for keywords related to IoT such as "IoT", "devices". In addition, reports on case studies of attacks that has been launched against IoT devices as well as attacks launched against outside network domain using IoT devices were reviewed [43][10]. The CVE IDs of the vulnerabilities that was exploited in the attack was collated, and we retrieved detailed information on the vulnerabilities from [42] using their CVE IDs.

- **Suggestions**

  This stores suggestions on how to fix known vulnerability in IoT. This table contains a foreign key (VulnerabilityID) that is referenced from the *Vulnerability* table.

- **Scanners**

  This stores information about vulnerability scanners integrated with the system.

Figure 5.3 shows the relationship between each entity in the database. *DeviceBios* stores the bio-data information and stores distinct ID for a scanned device. The Device ID (a primary key) in this table is used to reference the a specific device from the *ScanResults* table, and other entities (*MiraiReport*, *SshScannerReport*, *NmapScannerReport*, *DevicePortsUsed* and *DeviceOSInformation*)not shown in the diagram. The *DeviceBios* table also stores foreign keys *OperatingSystemID* and *EncryptionID* from *OperatingSystems* and *EncryptionAlgorithm* respectively. Data table *Vulnerability* stores known IoT vulnerability. Each vulnerability has a primary key Vulnerability ID referenced by *Suggestions* table. For each vulnerability, a remediation suggestion is stored in *Suggestions* table. SeeSec's database can be generated by executing a python script (*Database.py*) implemented for ease of deployment.

5.4 Vulnerability Scanners

We implemented and integrated four (4) vulnerability scanners for the purpose of this research.

5.4.1 Nessus Vulnerability Scanner

Nessus is a vulnerability scanner developed by Tenable Network Security, which is free for personal use. Nessus allows scans for vulnerabilities that allow a remote hacker to control or access sensitive data on a system, misconfiguration vulnerabilities (e.g. missing security patches), default password, and common password (dictionary attacks). Nessus provides varieties of plugin used for different scan types. SeeSec launches vulnerability scans by connecting to Nessus via REST API.

The scan report from Nessus is retrieved and stored in the file a .csv format which is subsequently parsed and stored in SeeSec's database. For the purpose of this research, we ran 4 types of vulnerability scan using the Nessus tool: badlock detection scan that checks for weak lock and insufficient authentication/authorization; host discovery used to discover live hosts and open ports; insecure network services and insecure software/-firmware; and DROWN detection that performs scan on SSL and TLS services on a device and checks for lack of transport encryption (Table 6.1.

5.4.2 SSH Scanner

This scanner checks if port 22 (used for secure shell communication protocol) of an IoT device is opened, and attempt a brute-force login with common usernames and passwords. Once it successfully obtains access to the IoT device, the user is alerted about the vulnerability and prompts the user for consent to fix (by changing the password) the vulnerability. The scanner returns the login status of the device scan, and stores the output in the database. The device is flagged 'non-vulnerable' if port 22 is closed, and flagged vulnerable if the port is opened and a login attempt was successful using a combination of common username and password. This scanner attempts to login to an opened port 22 using a total of 462 combinations of default/common usernames and

password.

The implementation of this scanner was inspired due to the recent increase of hackers using vulnerability in OpenSSH to attack Internet of Things (IoT) devices [44] and remotely generate traffic. Ory Segal and Ezra, researchers at Akamai technologies identified this vulnerability and claims hackers are using it to target CCTV, NVR and DVR devices, and networking devices. These devices are then subsequently used to mount attacks against internal networks that hosts them.

5.4.3 Nmap Scanner

Nmap is a free and open source network discovery and security auditing utility. It uses raw IP packets to determine what hosts (IoT devices) are available on the network, what services (application name and version) those devices/hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use as well as other characteristics.

The state of a port is either open, filtered, closed, or unfiltered. Open means that an application on the target machine is listening for connections/packets on that port. Filtered means that a firewall, filter, or other network obstacle is blocking the port so that Nmap cannot tell whether it is open or closed. Closed ports have no application listening on them, though they could open up at any time. Based on our system design, a closed port is labelled as non-vulnerable, and opened ports are flagged to be potentially vulnerable.

Ports are classified as unfiltered when they are responsive to Nmap's probes, but Nmap cannot determine whether they are open or closed. Nmap reports the state combinations open—filtered. and closed—filtered, when it cannot determine which of the two states describe a port. In addition to the Nmap commands, we used nmap scripting engine (NSE) scripts.

NSE is one of Nmap's most important and flexible features that allows users to write and share simple scripts to automate a wide variety of networking tasks. The scripts are executed in parallel and can be used for vulnerability detection and exploitation.

NSE scripts define list of categories they belong to; categories include auth, broadcast, exploit, version, vuln etc. For the purpose of this research, we used vuln category. The *vuln* category checks for specific known vulnerabilities and only reports if those vulnerabilities are found.

For the purpose of our implementation, we used NSE vulnerability script and the following command:

```
'nmap -O -sV -A --script vuln -oX {0} {1} --reason'.format(scan_result_name,
    IP_Address)
```

- **-O** - This is used to detect the operating system running on a device in the network. It sometimes guesses the operating sysem running on the device and gives an accuracy score on the operating system name it reports. This enable us to know how reliable the information provided can be used for further actions required to conclude on a vulnerability status of the device.

- **-sV** - This is used for service and version detection. Nmap probes open ports to determine service or version information used on the port. When Nmap discovers a port is opened, version detection interrogates the open ports to determine more about what is actually running. Nmap tries to determine the service protocol(e.g. FTP, SSH, Telnet, HTTP), the application name (e.g. Solaris telnetd, Apache httpd), the version number, host name, device type (e.g. webcam, printer), and the OS family (e.g. Windows, Linux).

- **-A** - This is a miscellaneous command that enables operating system detection, script scanning, and trace route. This command can be used in place of both -O and -sV, but we still included both commands (-O and -sV) in the commands run by Nmap because findings shows the scan returns less information when only -A command is used.

- **–script vuln** - This command checks for specific known vulnerabilities and only reports if those vulnerabilities are found. This script checks for vulnerabilities

against opened ports depending on the services running on that port. The vulnerability checks run includes, ssl-heartbleed (detects whether a device is vulnerable to the OpenSSL heartbleed bug), ssl-known-key (checks whether the SSL certificate used by a host has a fingerprint that matches an included database of problematic keys).

- **-oX** - This command is used to specific the output format to be returned by the Nmap scanner. For the purpose of this research, we used the xml file format. Nmap stores the result of the scan in the SeeSec's file server and the output is subsequently parsed and stored in the database.

5.4.4 Mirai Scanner

This scanner checks if an IoT device could easily be hijacked by attackers and used for malicious purposes. A free tool created by Rapid7 Metaspolit, called IoTSeeker, was integrated with SeeSec to check if an IoT device still has the default username and password (factory credentials).

IoTSeeker is used to detect default password vulnerability in an IoT device by establishing a TCP connection with the device. If the scanner is able to establish a connection, it checks the type of device it is, and subsequently try to log into the device using the factory credentials associated with that device type. The implementation of this scanner was inspired by the recent distributed denial of service (DDoS) attack launched against Dyn [45], a domain name server (DNS) provider, and Krebsecurity [46], an in-depth security news and investigation website, that left many of the web's most popular sites unreachable.

IoTSeeker uses the perl module *AnyEvent* for high parallelism, hence, it was required to install perl on the host machine of our SDN controller to run and successfully integrate this scanner.

Chapter 6

EVALUATION RESULTS

Evaluation of the implemented security framework is categorized into two: simulation of virtual IoT networks, and evaluation using human subjects and their personal device(s) (mobile phone, electronic tablet and laptops).

6.1 Virtual IoT Network Simulation

After implementing the required modules, we ran tests to ensure the framework functions as intended by simulating a virtual IoT network using Mininet [47] . Mininet is an open source network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts runs standard Linux network software, and its switches support OpenFlow for highly flexible custom routing, and Software-Defined Networking.

We configured a Mininet virtual machine running a bridged network adapter that allows external (Internet) communication. We remotely login to the virtual machine via SSH, which allow us to establish multiple remote connection instances. One instance (window) is used to manage the POX controller and the second is used to manage the virtual network of hosts (IoT devices).

With the implemented system configured in the POX controller, we simulated two scenarios of IoT network by setting up a logical test bed of 3 hosts (IoT devices) that do not have assigned IP address, and 6 hosts with static IP addresses, connected to an OpenFlow (OF) switch. Each of the simulated network consists of a remote controller, POX, and hosts (IoT devices) connected to an OF switch. The remote controller manages the entire network, and controls the activities of the OF switch.

We launch the IoT network by executing a python script, *dhcpmininet.py*, and start the POX remote controller by running *pox.py*, and a start up script, *startup.py* that

automatically launch other components required to test the implemented framework. The POX components used for our test include the DHCP server (*dhcpd.py*) that handles the DHCP requests, forwarding L2-switch (*l2_forwarding.py*) that handles packet forwarding, a firewall component (*firewall.py*) that installs communication rule, host tracker (*host_tracker.py*) for tracking scanned hosts, and the OpenFlow component (*discovery.py*) that serves as the switch for the network and listens for connection from the remote controller, POX.

For the first scenario, the simulated IoT network is a single topology network of 3-hosts that does not have assigned IP addresses, 1-OF-switch, and a remote controller, POX. DHCP request was sent to the DHCP server for each hosts to get assigned IP addresses. The DHCP server was configured to lease IP addresses ranging from 192.168.0.13 to 192.168.0.253, with the network's default gateway IP address as 192.168.0.254.

SeeSec launches vulnerability scans on an event (*DHCPLease* event) the DHCP server leases an IP address to a host in the network. The result of the scan (if an IoT device is vulnerable or not) determines the rules the firewall component insert into the OF switch through the POX controller. The controller subsequently updates the flow table entry of the OF switch. For each vulnerability scan, the network policy (firewall rules) is updated, this is done by remote method invocation. The method *updateList(list_type, mac_address)*, which is invoked via the scan server interface takes in *list_type* and *mac_address* as parameter values, where list_type could be blacklist or whitelist.

For each scenario, we configured the scan server to run different types of vulnerability scans (Table 6.1) with different scanning tools (Nessus vulnerability scan and SSH scanner). Table 6.1 shows the functions and the type of vulnerabilities that can be detected by each scan type. The scan server analyze the results and flags a device vulnerable if more than 5 percent of the results obtained from the integrated scanning tools flagged the device vulnerable.

According to the results obtained from the first scenario, one of the hosts (h2) was deduced to be vulnerable and the other hosts (h1 and h3) were deduced to be safe based

Table 6.1: Types of Scan

Scan Type	**Description**	**Vulnerability**
Basic Network Scan	Performs a full system and network scan in a host	Insufficient security configuration
Badlock Detection	Performs checks for the weak lock vulnerability	Insufficient authentication/authorization
DROWN Detection	Performs scan on SSL and TLS services on the host	Lack of transport encryption
Host Discovery	Performs simple scan to discover live hosts and open ports	Insecure network services, insecure software/firmware
Custom scanner (SSH scanner)	Performs checks for common weak/default password vulnerability	Insufficient authentication/authorization

on the analysis done by the scan server. The ACL and firewall policy was subsequently updated accordingly. Packets that were sent to and from the vulnerable host (h2) in the network were dropped, which confirms that any form of communication was disabled in the vulnerable host. The functionality of enabling and disabling communication access in an IoT device is executed by the firewall component. Figure 6.1 shows communication was disabled based on the firewall policy.

In the second scenario, the simulated IoT network is a single topology network of 6-hosts with preassigned (static) IP address, 1-OF-switch, and a remote controller, POX. This scenario is to test the host tracker component of our framework functions as intended. Packets were sent at random between hosts in the network. For every packets sent between hosts, the host tracker component checks if both source and destination host has been scanned for vulnerabilities by checking the ACL. Vulnerability scans were launched for unscanned hosts, and the firewall inserted rules in the controller based on the scan results from the scan server.

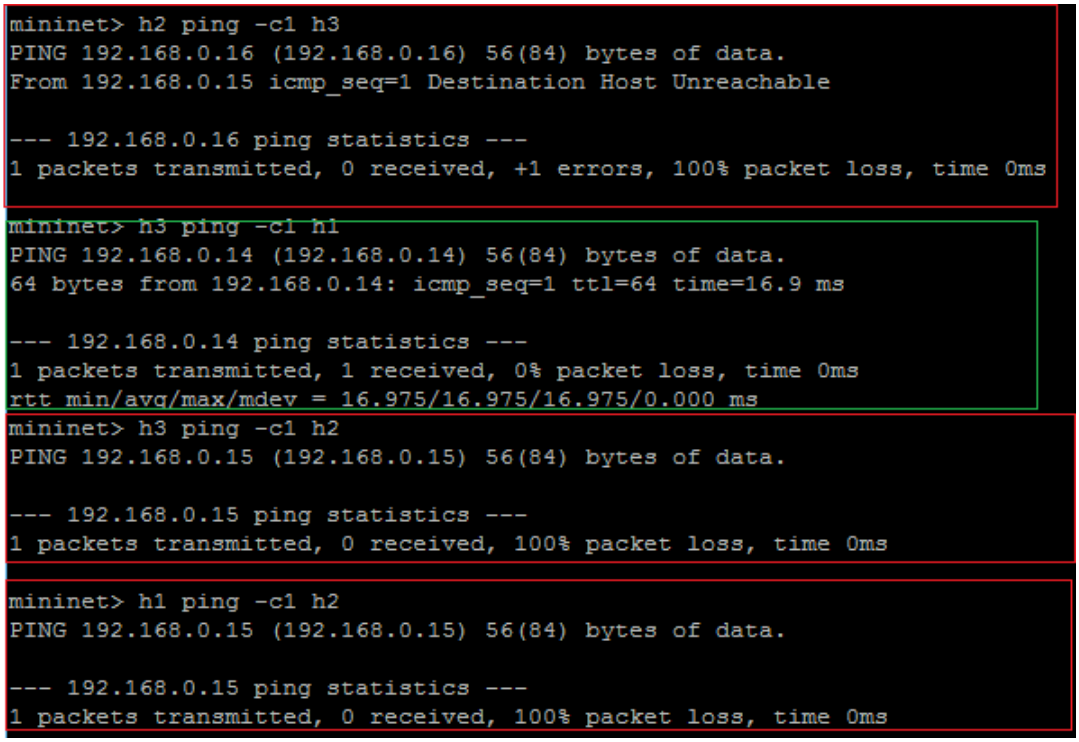

Figure 6.1: Communication flow between Blacklisted and Whitelisted Hosts

6.1.1 Response Time

To test the impact of running an extra check to confirm if a device/host have been scanned or not, we compared the time it takes to send and receive packets between hosts via the controller and without the controller. Figure 6.2 shows the average response time to send and receive packets between hosts in the simulated network. The average response time when packets are sent via the controller is 31.7ms (milliseconds) and 26.7ms without the controller resulting in difference of 5.1ms. Mininet implemented openflow switch may take tens of milliseconds to process incoming packets, due to other high-priority processes. This delay affects the response times computed by ping technologies, because the reply test packets might be sitting on queue while waiting to be processed. Therefore the response times would not accurately represent true network delays. However, compared to most commonly used response time threshold of 30ms,

the response time when packets are sent via controller can be graded as average in terms of network performance.

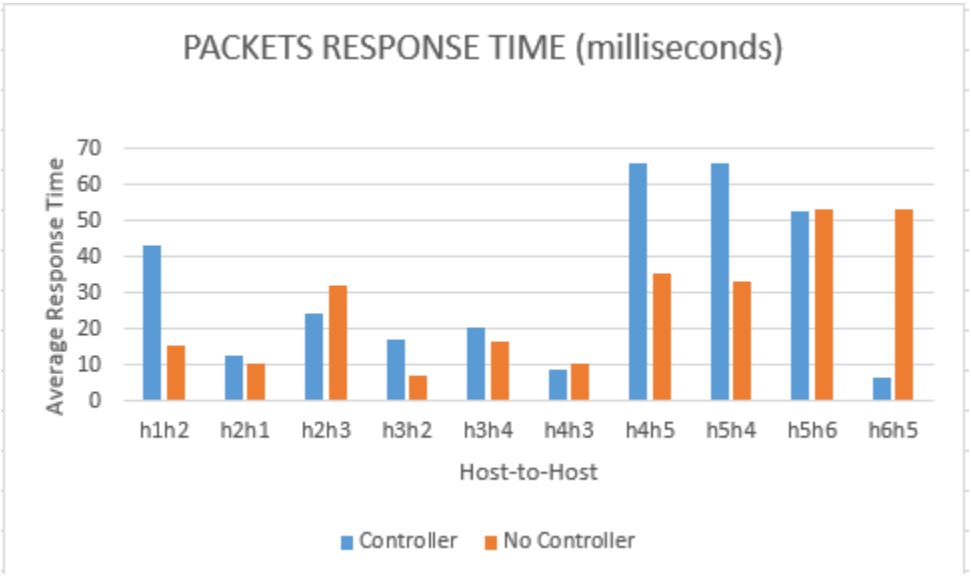

Figure 6.2: Average Response Time to Send and Receive packets between Hosts

6.1.2 Bandwidth

Figure 6.3 shows the percentage usage of the network bandwidth with and without a controller. The average percentage bandwidth usage when packets were sent via the controller and without the controller is 90.75% and 91.15% respectively, a difference of 0.45% which is negligible.

6.1.3 Scan Time

For the implemented custom scanner that detects common weak/default username and password, port 22 in a host is checked to determine whether the port is opened or closed before running combinations of username and password. A total of 462 weak/default usernames and passwords combinations were run against an open port, as devices with closed ports are flagged non-vulnerable. It takes an average time of 1698.35 seconds to run all 462 combinations, resulting to an average of 3.676 seconds for one combination.

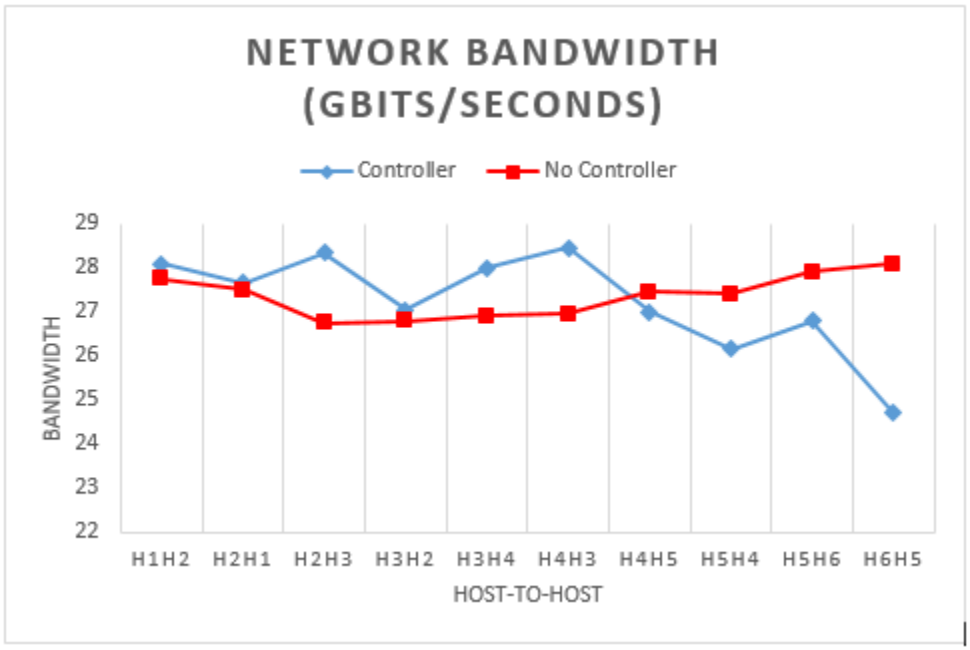

Figure 6.3: Network Bandwidth between Hosts

Since we ran the test on a simulated network, we confirmed this functionality works by assigning the IP address of the host machine (a VM) to one of the host in the network, and configured the VM's login credentials to be one of the username/password combinations. Given user's consent, the system changed the password on a successful login using a weak/default username and password combination, and generated a report on the fix done, including the newly generated password. The new password is generated by random combination of alphanumeric characters of length eight. It takes an average of 0.646 seconds to resolve a weak/default password vulnerability.

As shown in figure 6.4, it takes an average of 2.12 minutes to scan for weak/default password using a custom scanner, which is significantly less than the time (7.25 minutes) it takes the Nessus penetration testing tool to run a badlock detection scan that scans for the same type of vulnerability as the custom scanner (SshScanner). This shows the feasibility of detecting vulnerabilities in a reduced amount of time if we implement only custom scanners that check for IoT-specific vulnerabilities. The scan time can be

reduced further by parallelizing processes run by the scanner.

Figure 6.4 shows the overall average scan time using Nessus vulnerability scanner is relatively high. The time it takes to complete a vulnerability scan on a device is dependent on the scanning tool used, because they use different approach to detect vulnerabilities. In addition, the scanning tools scans the entire network for all possible vulnerabilities including the vulnerabilities that are not IoT specific.

While the delay on these scans will be noticeable by the user, we argue that the delay on the order of minutes is worth the additional security benefit since it is a one time cost for a recurring benefit. However, it is necessary to develop a mechanism that reduces the scanning time to enable a near real time network access to devices that are not vulnerable. Using custom scanners will help to reduce the scan time significantly as it will only check for IoT specific vulnerabilities.

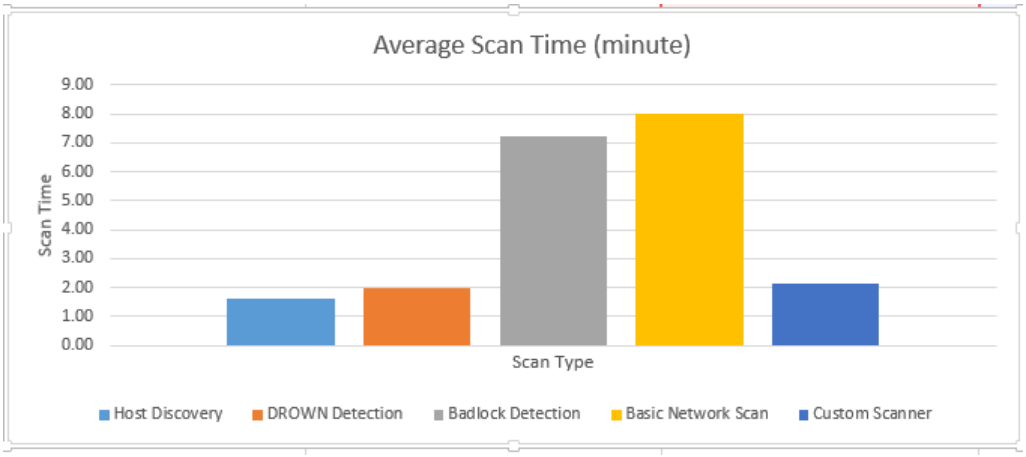

Figure 6.4: Average Scan Time (Virtual Network Simulation)

6.2 System Testing Using Human Subjects

We evaluated the feasibility of the implemented security framework using real IoT devices and OpenFlow-enabled switch. As shown in Figure 6.5, we configured the HP2920 POE+Switch to enable OpenFlow protocol. In addition, we set up a Linux machine that host the POX controller (See details of configuring HP2920 as an openflow

switch subsection 6.2.4). Participants connects their smart device (mobile phone and laptops) to the private network set up via Ruckus wireless access point.

6.2.1 Evaluation Goals

The primary goal of this experiment is to evaluate the usability of the system, quality of service (in terms of suggestions on how to fix vulnerabilities), user interaction, and the overall stakeholders' satisfaction. To validate SeeSec quality of service, we define the following evaluation goals and methods.

1. System usability.

   One of the goals of this research is to enable non-technical users (and technical users) to set up a secured IoT network. To test the usability of the system, human subjects were used in this experiment. The purpose of this is to measure the quality of the content in the suggestions provided by the system for fixing unresolved vulnerability. In addition, the experiment will help to identify the ease of adopting SeeSec in terms of user's interaction. Data used for to analyse the system usability were obtained from human subjects (participants).

   - **Participants**

     The experiment consisted of 28 participants from different fields and major at the University of Washington Bothell. To achieve more efficient results and eliminate redundancy in scan results, participants were encouraged to use their personal IoT device(s). Each participant used their personal smart device(s) which include smart phone, laptops and electronic tablets. Participants included both technical and non-technical personnel to ensure the system is usable by SeeSec's primary stakeholders (non-technical users).

   - **Questionnaire**

     To measure the quality of the suggestions provided by SeeSec, participants were instructed to fill a questionnaire. It was ensured that the questions that will give insight to the participants satisfaction and experience were asked.

Examples of questions asked include: rate how easy it was understand remediation suggestions provided, how fast it was to join their device to the network etc. The Likert scale was used to enable easy evaluation of participants' attitudes about the usability of the implemented framework. In addition, Likert scale helps to easily quantify participants' overall satisfaction, and to measure the quality of service.

- **Interview**

  Semi-structured interviews were conducted with each participant, and responses were stored via audio recording. The purpose of the interview was to get participants' opinion about the security framework.

6.2.2 Experimental Setup

This section explains the structure of the experiment test bed. To ensure SeeSec functions as intended, a real IoT network that consists of an OpenFlow-enabled switch (Aruba HP2920 POE+Switch), connected to a remote controller, POX which was hosted on a Linux machine. Figure 6.5 shows a graphical representation of the network test bed, and the communication flow between components of the network.

The experimental setup allows only one participant to carry out the test at a time (only one screen display is available at a time). To enable user's awareness about the events occurring in their IoT device, vulnerability scan information is rendered on a display screen as the events occurs. Another purpose of the display screen is to enable user interaction with the system: users can give consent to SeeSec to run automatic update, change password etc.

6.2.3 Experiment Procedure

1. **Participant registration and orientation**: Participants register and sign the consent form. Participants were informed about the process flow of the entire experiment, and were also informed on what they will be required to do after the system testing is complete, i.e. fill out the questionnaire and an interview with

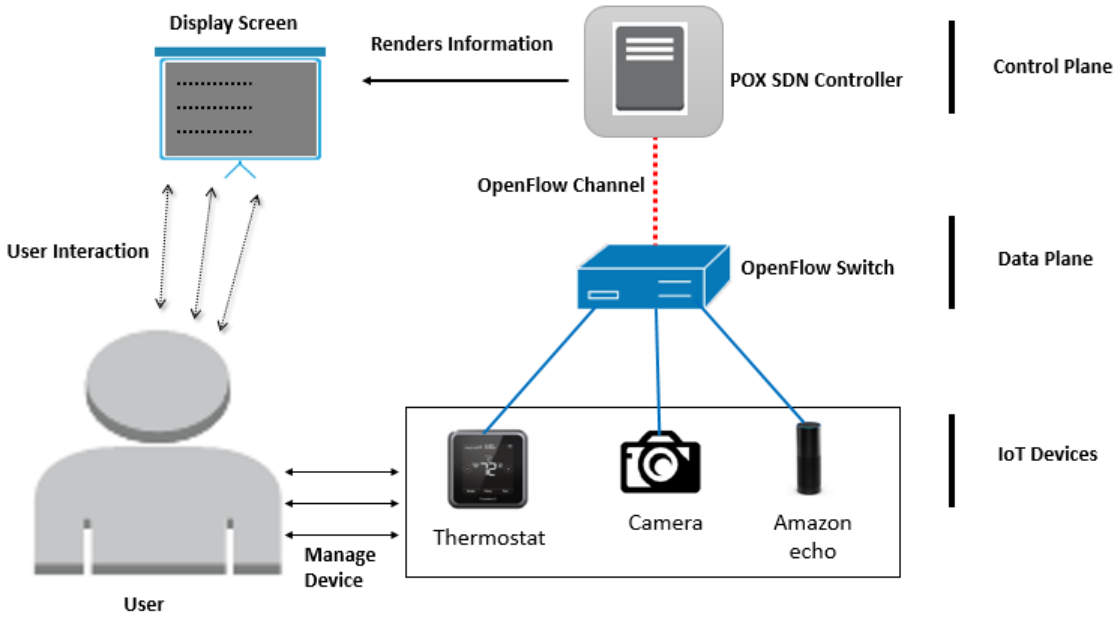

Figure 6.5: Diagram showing the Experiment Test Setup

the researcher.

2. **Device installation**: Participants install their smart device(s) by connecting to the IoT network through a wireless access point (Ruckus WAP).

3. **User input and Installation feedback**: During installation, information on installation progress and scan results is provided to the participant. This information is displayed on a screen, and when necessary (when prompted by the system), participants give their consent to the system to fix detected vulnerabilities. The system resolve vulnerabilities provided users gives their consent, and suggestions on fixing unresolved vulnerabilities is provided to the user.

4. **Questionnaire**: Participants fill the questionnaire immediately after completing the system test. Participants were required to read through the scan report, and suggestions on fixing unresolved vulnerabilities (for vulnerable devices) in their

IoT devices before they fill the questionnaire. This is to ensure participants provide valid feedback based on their actual experience with the security framework.

5. **Interview**: Researcher interviews participants using semi-structured questions.

6.2.4 Configuring HP2920 as OpenFlow-enabled Switch

The HP 2920 is optimized for Software-defined Networking (SDN) with OpenFlow support, and is designed to be used primarily as a desktop switch to which end nodes, printers and other peripherals, and servers are directly connected to. Other examples include wireless access point (WAP), PCs and peripherals server etc.

The switch was configured to act as a layer-2 (L2) switch, which enables virtual local area network (VLAN) support and tagging, and automatic learning and dynamic assignment of VLANS. A virtual LAN or "VLAN" is a logical subdivision of a Layer 2 network that makes a single Layer 2 infrastructure operate as though it were multiple, separate Layer 2 networks.

To configure the switch, we connected a console (a PC) to the switch using the Micro USB console port on the switch. For the experiment, we created four VLANs: VLAN1 which is the default VLAN used for switch management and configuration, VLAN2 is the controller VLAN used as an interface for the remote POX controller, VLAN3 and VLAN4 are the openflow VLANs with untagged ports 6-19 as shown in listings 6.2.4. In other words, the POX controller manages the openflow VLANs, and the devices that connects to the VLANs via the untagged ports lease IP addresses from the DCHP server of the POX controller.

```
HP2920 Running configuration:
 openflow
   controller-id 1 ip {ip_address} controller-interface vlan 2
   instance "openflow"
       listen-port
       member vlan 3, 4
       controller-id 1
```

```
        enable
        exit
    enable
    exit
oobm
    ip address dhcp-bootp
    exit
vlan 1
    name "DEFAULT_VLAN"
    no untagged 6-24
    untagged 1-4
    tagged 5
    ip address dhcp-bootp
    ip helper-address 192.168.240.254
    exit
vlan 2
    name "ControllerVLAN"
    untagged 20-22
    ip address 69.91.2.2 255.255.255.0
    ip helper-address 192.168.2.1
    exit
vlan 3
    name "OpenFlowVLAN1"
    untagged 6-15
    ip address 69.91.3.2 255.255.255.0
    exit
vlan 4
    name "OpenFlowVLAN2"
    untagged 16-19
    ip address 69.91.4.2 255.255.255.0
    exit
```

We enabled openflow on the switch, created an openflow instance called "Openflow"

in a virtualization mode, and set the properties of the openflow instance accordingly as shown in listing 6.2.4. The created openflow instance context, *openflow*, is independent and has its own openflow configuration and controller connection. VLAN 3 and 4 were designated as members of *openflow*. OpenFlow was set to work in active mode i.e. new packets of a flow that the switch isn't aware of are sent to the OpenFlow controller.

In order to associate *openflow* with the remote POX controller, we configured a controller instance and specified the VLAN to be used as its interface. Up to three controllers can be associated with an openflow instance, which can enable us deploy SeeSec using distributed SDN controllers.

```
Configured OF Version     : 1.0
Instance Name             : openflow
Administrator Status      : Enabled
Member List               : VLAN 3, 4
Listen Port               : 6633
Operational Status        : Up
Datapath ID               : 00039cdc71f43180
Mode                      : Active
Flow Location             : Hardware and Software
Conn. Interrupt Model     : Fail-Secure
Maximum Backoff Interval  : 60 seconds
Probe Interval            : 10 seconds
Hardware Table Miss Count : 0
No. of Software Flow Tables : NA
Table Model               : Single Table

Controller Id   Connection Status  Connection State  Secure   Role
--------------  ----------------   ------------------ -------- ------
1               Connected          Active            No       Equal
```

6.2.5 Stake holders Technical Expertise and Security Concerns

The primary stakeholders of the implemented framework are non-technical consumers of IoT. According to the survey carried out during the experiment, 78% of the participants were enrolled in non-computing related majors (e.g. Cultural Studies, Health Studies, Biology, Psychology etc.) and 26% in computing related field (e.g. Computer Science, and Applied Computing etc.). The participants distribution across the two categories of field (computing and non-computing related) allowed us to get feedback from participants that might have been exposed to skills on how to setup a secured network or have more awareness on network security, and also participants that have little or no exposure to network configuration.

We conducted a semi-structure interview to get insight on the participants view on network security, and the actions they take to ensure their device(s) is not at risk of security breach. 23% of the participants claimed they have excellent skills to set up a network of IoT devices but none of them claimed to have an excellent skills in setting up a secured network of devices. About 87% of the participants own at least one smart device, and 44% having at least 3 IoT device, which shows IoT is truly becoming an integral part of our daily lives.

While 22% of the participants had no knowledge about network security, 63% of the participants' understanding of network security is to have privacy and also to ensure they change their password, and 4% do not care about security regardless of being a consumer of IoT. Despite the lack of knowledge on network security, there was a great deal concern about security (Figure 6.6).

93% express concerns about their device being part of a botnet used to launch a cyber attack. Of the 93%, 8% are worried that an intruder can have access to sensitive data that could be for profiling, stalking or track their activities (social engineering), 52% are concerned about potential hacking, privacy breaches, and data theft carried out against their connected device, 8% are concerned about legal sanctions, and 12% were worried about authorized access whereby an attacker could have full control of their device.

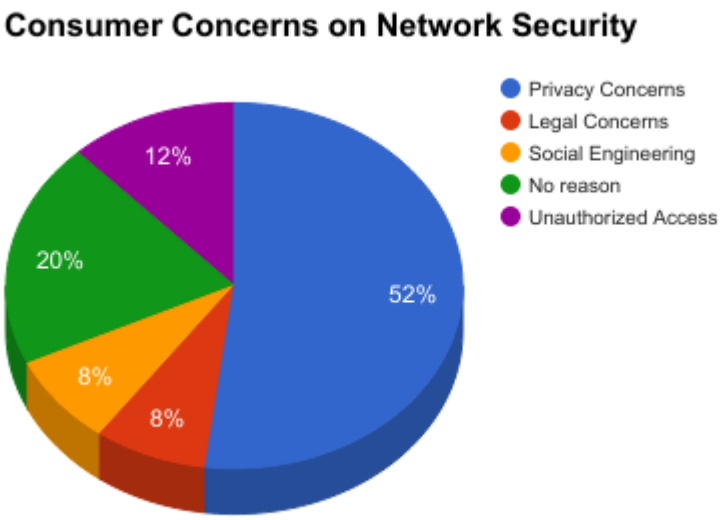

Figure 6.6: Consumers' Concerns on Network Security

Despite the high level of concern, 14% of the participants never change their password, while 46% change their password only when prompted. In addition, majority of the participants does not update their device's firmware when they receive notification to do so. Reasons for not updating their firmware include: concerns that their smart device might crash after an update, waiting for reviews from colleagues/friends, and lack of time to read the details of the new update of a firmware.

6.2.6 System Usability

The system usability evaluation focuses on how well users can learn and use the implemented framework to achieve their security goals. We aimed to measure the usability in three categories: effectiveness (can the participants successfully achieve the security goals SeeSec intend to provide), efficiency (how much effort and resource is expended in achieving those objectives) and overall satisfaction (was the experience satisfactory).

Table 6.2: Five-level Likert Item Response and its Equivalent value

Response	**Value Equivalent**
Strongly Disagree	1
Somewhat Disagree	2
Neutral	3
Somewhat Agree	4
Strongly Agree	5

Participants were asked to select their level of agreement or disagreement on assertive statements that describes qualities of the system in terms of usability.

Five ordered response levels were used on each Likert items (assertive statements), and the responses were translated to the equivalent values shown in Table 6.2 for easy analysis.

Table 6.3 shows majority (87%) of the participants accepted it is easy and quick to connect to the network; 59% strongly agreed, 28% somewhat agreed and 7% stayed neutral. The results from the survey also shows participants are willing to adopt and purchase the security service SeeSec provides.

Survey data shows 49% of participants will adopt the service, and 31% on a neutral stance (made no choice whether they will adopt the service or not). Further analysis on the high percentage of participants on neutral stance shows there are dependence factors to which some participants would agree to adopt a security service. Interview data shows 39% of the participants will only purchase the service if it affordable or free. Of the participants (46%) that are willing to purchase a security service, 54% prefers a one-time cost security service, while the others are satisfied with a monthly subscription service plan.

Based on our findings, a significant percentage (76%) of the participants believe they need to learn a lot or need technical support to use SeeSec. We believe participants' understanding of this question is that they will need to learn about network security,

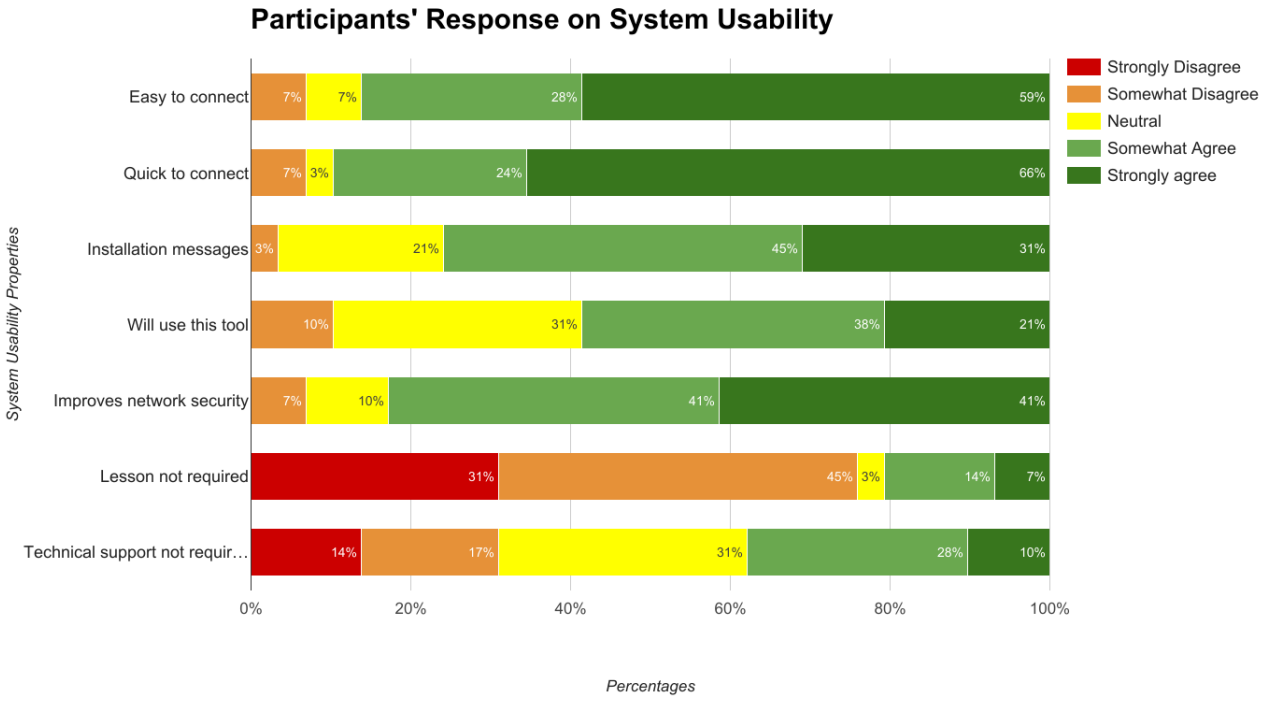

Figure 6.7: Percentage Distribution of Participants' Feedback on System Usability

this is because only 31% of the participants disagreed technical support is not required to use SeeSec, and 31% neutral. Because of the lack of correlation between the responses between the related likert item, we conclude that the reliability of the data (for the two likert item: lesson not required, and technical support not required) is insufficient to derive a valid conclusion. Majority (82%) of the participants were satisfied with the system in terms of trusting the system to improve their network security.

To compute system usability score, each response from the likert scale was converted into an equivalent value as shown in Table 6.2. We computed an overall system usability score of 74% using the formula:

$$Z = \frac{\sum_{i=1}^{n}(\sum_{j=1}^{m} x_j)_i}{K * m * n} \tag{6.1}$$

where Z is the system usability score, n is the number of question items, m is the number of participants, and K is the most positive response value which is 5 in this case. Based on research [48] using the system usability scale originally created by John Brooke, a system usability score above 68 is considered above average. Our derived normalized score of 74 significantly greater than the threshold, hence it can be concluded that the performance of SeeSec in terms of system usability is in third percentile ranking.

6.2.7 User Interaction

We used semantic differential scale to evaluate SeeSec's user interaction design. Semantic differential scales are used to gather data and interpret data based on the connotative meaning of the respondent's answer. To determine the quality of the system's user interaction, we analyzed participant's rating on characteristics that interprets the quality of SeeSec's user interaction design. Each characteristics contained two dichotomous words at either end of the spectrum, to enable us to measure more specific attitudinal responses as shown in Table 6.3. In other words, the responses are broken down into factors: negative and positive.

The rating of each user interaction design questions was scaled from 0 to 9, with 0 being the lowest (most negative response) and 9 being the highest (most positive response) score. Using a scale from 0 to 9 enabled use to easily compute the score for each characteristic by multiplying each ratings by the number of participants that gave that specific rating.

As shown in figure 6.10, the system scored over 80% in terms of: connection speed, prompts for user input, and organization of information. While the score for consis-

Table 6.3: SeeSec's User Interaction Design Ratings

Feature quality	**Negative**	**Positive**	**Score (%)**
Interacting with the system (entering inputs)	Difficult	Easy	84
Use of terms throughout installation	Inconsistent	Consistent	81
Prompts for input	Confusing	Clear	80
Organization of information	Confusing	Clear	75
Tools inform about its progress	Never	Always	74
Error messages	Unhelpful	Helpful	73
Speed of connecting to the network	Too slow	Fast enough	72
Designed for all level of users	Never	Always	60

tency in the use of terms throughout installation and progress report is 75% and 73% respectively, 36% participants think the system is not designed for all level of users (the participants gave a rating between 0 and 4).

From the analysis done on the data gathered from participant's interview, we identified the communication channels that can be used to provide information to the users. Figure 6.9 shows percentage distribution of communication channel preferences i.e. channels in which users would prefer view scan reports, receive remediation suggestion or interact with the user to give consent to automatically resolve vulnerabilities. The most preferred communication channel was via Email, 61% of the participants ranked this channel as their first preference. Table 6.4 shows a detail statistics of the communication channels preferences.

6.2.8 Vulnerability Resolution

To evaluate the content quality of the scan report and the suggestions provided for fixing unresolved vulnerability, participants were required to state how much they agree or disagree to statements that explains the goal of the system's function in terms of

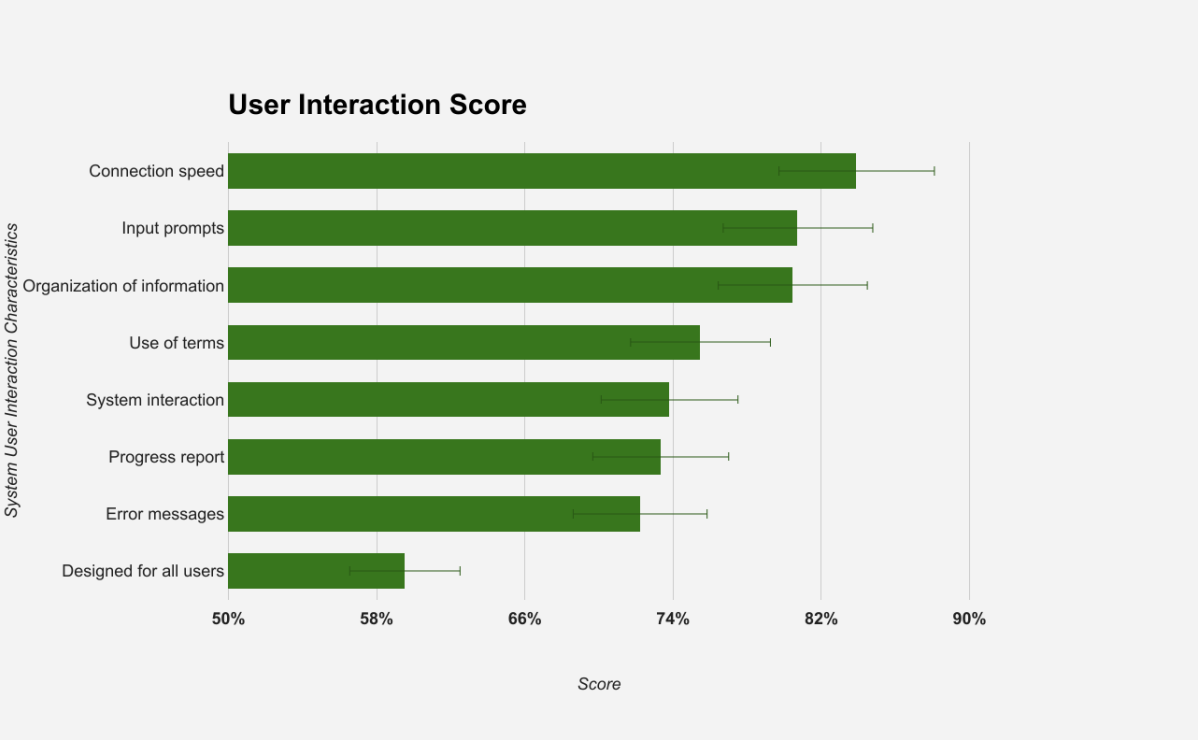

Figure 6.8: System User Interaction Evaluation Scores

automatically resolving vulnerabilities. In addition, we also asked questions that allows us to have insight on participant's opinion about automatic vulnerability resolution without requesting consent from users.

Figure 6.10 shows percentage distribution on how participants agree or disagree to the likert items used for evaluation (Table 6.5). Although the majority of participants (56%) disagree to vulnerability resolution without user's consent, 70% percent of the participants were agreed to vulnerability resolution without consent, if the resolution process does not breach their privacy or put them at risk (agreed to automatic software or firmware update).

The survey questions that measures quality of remediation suggestion and how easy it is to understand will be nullified for the purpose of this research. This is because there was no vulnerability found in any of the devices scanned during the experiment. We conclude that the participants that provided feedback on this item gave their rating

Table 6.4: Participants Communication Channel Preference

Communication Channel	**Preference(%)**	**Occurrence(%)**
Phone notification	8	12
Email	62	40
Mobile Application	23	25
Text Message	4	12
Web Interface	4	8
Voice mail	0	2
Desktop	0	2

Table 6.5: Likert items used for Evaluating System Resolution Design

S/N	**Likert Item**
1.	The suggestions provided to fix vulnerabilities in my device is easy to understand
2.	I can resolve the vulnerabilities based on the suggestions provided
3.	The scan report was presented in a clear format
4.	The scan report was easy to understand
5.	The system can resolve vulnerabilities on my behalf without my consent
6.	I prefer to give permission to the system before it resolves vulnerabilities, but it is okay to resolve vulnerabilities that do not breach my privacy or put me at risk

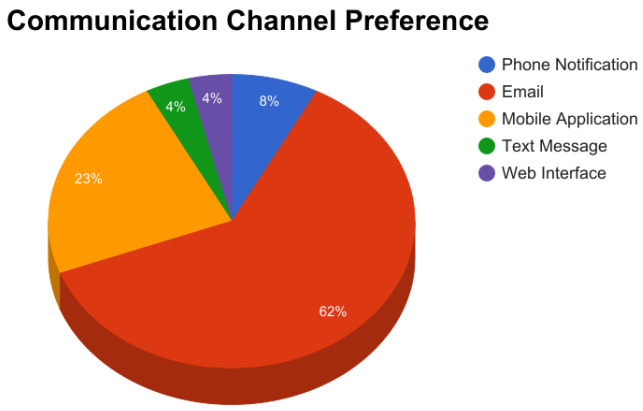

Figure 6.9: Percentage Distribution of Participants Communication Channel Preference

based on the verbal discussion they had with the researcher during the experiment.

Based on participants response on the scan report format and how easy it is to understand the scan report, 61% agreed the scan report is well formatted. This result shows a slight correlation with the participants' feedback (a score of 75% for organization of information) from the user interaction evaluation (Figure 6.8).

Only 26% agreed the scan report is easy to understand, and 29% were unable to agree or disagree. During the interview sessions, some participants expressed their concerns about their inability to understand some of the terms used in the scan report. Although there was a positive feedback (81%) on the use of terms throughout installation (Table 6.3), findings from the interview data shows it is necessary to enhance the quality of data in the scan report to enable ease of understanding.

6.2.9 Scan Time

For the experiment conducted with human subjects, SeeSec was configured to run only three types of vulnerability scanners: Nmap scanner, SSH scanner, and Mirai scanner.

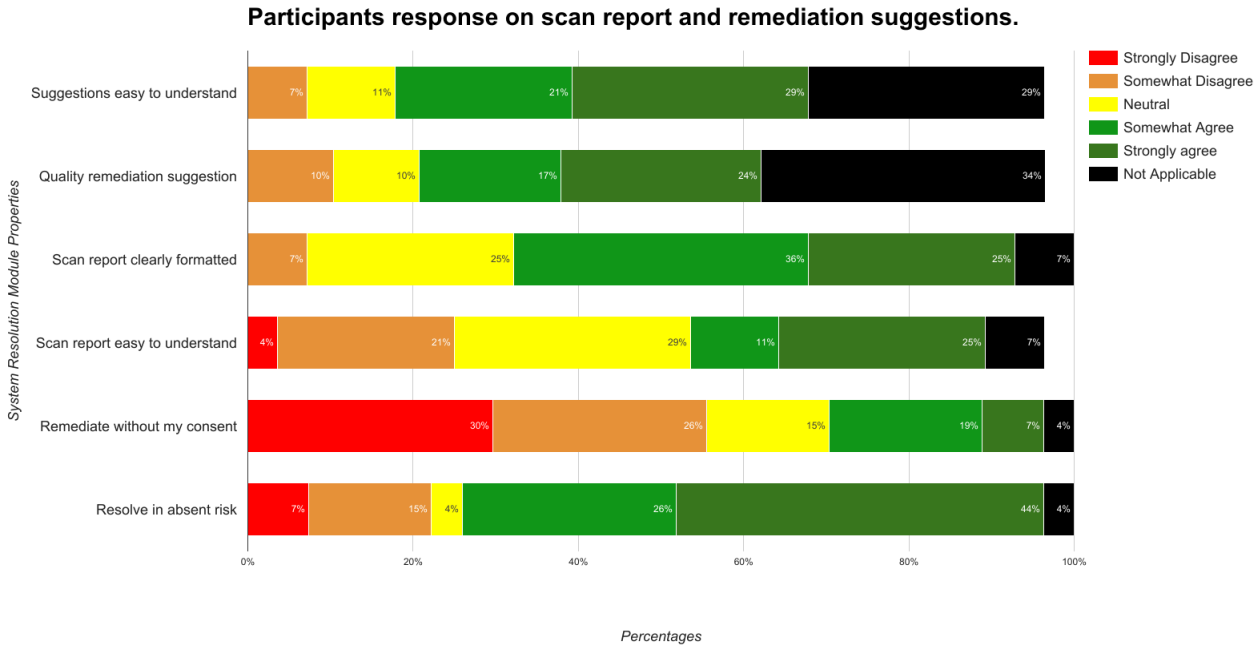

Figure 6.10: Percentage Distribution of Participants' Response on Scan Report, and Suggestions provided for Resolving Vulnerabilities

This is due to the average time it takes the Nessus vulnerability scanner to complete a scan, which ranges from 7 to 15 minutes. We stored the total scan time for each scanner for analysis. In addition, we compared the total time (aggregate scan time of all vulnerability scanners) it takes to complete a vulnerability scan an IoT device as well as granting communication access to a non-vulnerable device.

- **Nmap Scanner**

  The Nmap scanner checks for open ports, and tries to detect the services running on the open ports. Vulnerability checks are run against an opened port based on the services running on that port. It also try to tries to retrieve information about operating system versions of the device. The average scan time for the

Nmap scanner is 205 seconds (about 3.42 minutes). Based on our findings, it takes more time to complete nmap vulnerability scan in an IoT device that has one or more opened port. This is because the scanner tries to run vulnerability exploits against an opened port based on the service that the program perceive is running on that ports. Figure 6.11 shows the scan time when a device has all ports closed and when a device has at least one port opened. As shown in figure 6.11, there is a large difference between scan times, which lead to a significant increase in the average scan time for the Nmap scanner.

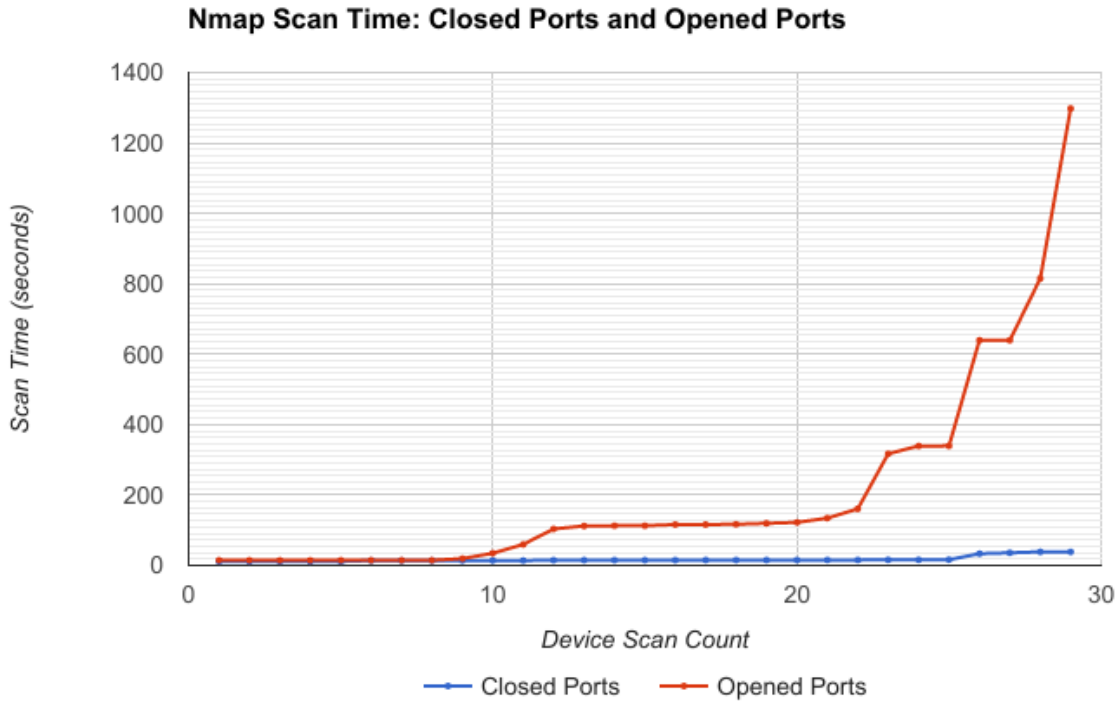

Figure 6.11: Nmap Scan Time for Devices with one or more Opened port versus All-closed ports

- **SSH Scanner**

  The SSH scanner first checks if port 22 is opened before trying to login into the

Table 6.6: Scan Time Overview (SSH Scanner)

Status	**Device**	**Scan Time(s)**	**Average Time(s)**	**#Combination**
Opened	7	27.93	3.99	10
Closed	69	1058.39	15.34	0
	76	**1086.31**	**14.29**	**10**

device using combinations of common default username and password. A total of 74 devices were scanned during the experiment, and scanned device types include mobile phone, laptops, and tablets.

The SSH scanner first checks if port 22 is opened before trying to login into the device using combinations of common weak/default usernames and passwords. A total of 74 devices were scanned during the experiment. Types of devices scanned include participants mobile phone and laptops, virtual machine, and devices on the researcher's network (netgear router,).

As shown in Table 6.6, the average time it takes to check if a port is opened is 15.34 seconds. The average time is as high as 15 seconds due to a loss of network connection while scanning eight of the devices. In other words, we could regards the scan time of the eight devices as outliers. The devices lost network access (the IP address assigned by the DHCP server expired), which resulted in the scanner probing an inactive IP address. Excluding the aggregate scan time for the eight hosts, the average time it takes to detect if a port is opened is 0.27 seconds. In addition, the average scan time to remotely login into a device using one combination of username and password is 2.79 seconds, which is tolerable in terms of connection speed.

- **Mirai Scanner**

  The scanner checks the type of IoT device, and tries to log in using factory credentials. The average time it takes to complete mirai vulnerability scan is 20.25

seconds. The mirai scanner attempts to create a TCP connection with a device to determine the device type, and then try to login into the device using its known factory credentials. The median time it takes to complete the mirai scan is about 0.20 second, which mostly due to the fact that the scan is aborted if the device type cannot be determined.

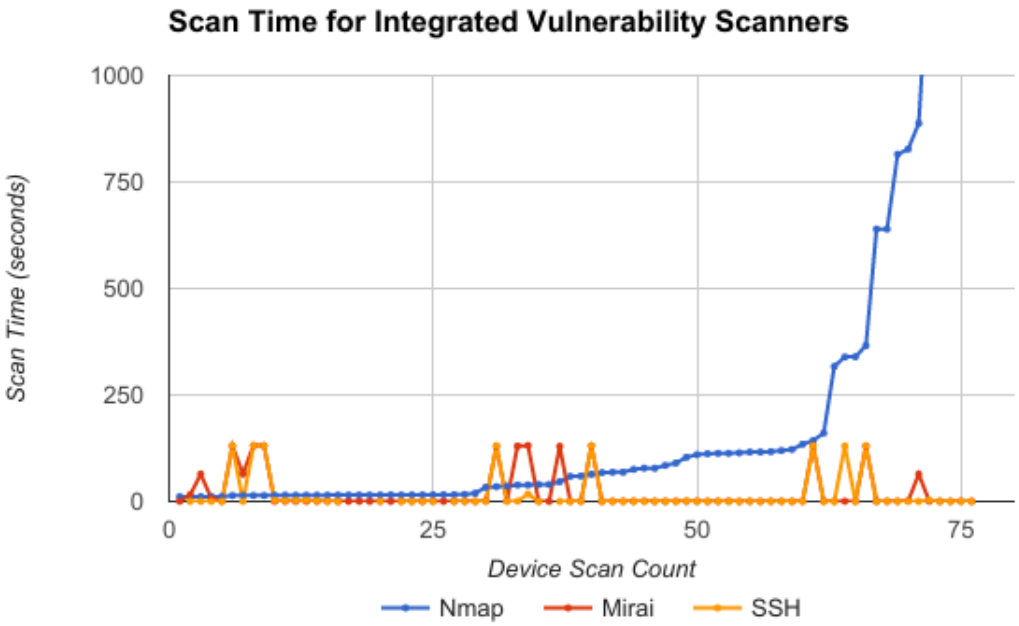

Figure 6.12: Average Aggregate Scan Time for the Integrated Vulnerability Scanners

The average aggregate time it takes to completely scan for vulnerabilities (using all integrated vulnerability scanners) in an IoT device is 256.36 seconds (4.27 minutes). This is relatively high considering the goal of joining an IoT device into the network near real time. As shown in Figure 6.12, the Nmap scan time is high compared to the other scanners (mirai and ssh). Since the Nmap scanner keeps probing an opened port for connection without a time limit, setting a limit to the time spent on probing an opened port could significantly reduce the scan time (a trade off to this is vulnerability detection rate). Although, the average time to complete vulnerability scans on an IoT

device is in the order of minutes, our results shows the feasibility of the implemented framework.

Chapter 7

CONCLUSION AND FUTURE WORK

We have implemented a security framework, SeeSec, that is a first step in providng security to consumers who lack the technical expertise to configure poorly secured and difficult to use IoT devices. Previous work has focused on security techniques that require manufacturers and administrators to build secure devices and systems. This research approach focuses on detecting flaws, automatically correcting them, and suggesting fixes when the system cannot automatically do so.

The results shows SeeSec achieved the design goals: scans for vulnerabilities in IoT devices using integrated vulnerability scanners, resolve vulnerabilities on behalf of users provided they grant the system permission, and provides suggestions on how to fix vulnerabilities.

The communication control (firewall) functionality of the system works as intended as show in Figure 6.1, whereby bidirectional communication to and from vulnerable devices is disabled. SeeSec is scalable whereby new vulnerability scanners can be easily integrated; it was straight forward to integrate new vulnerability scanners (Nmap and Mirai scanner) to the scanners implemented at the alpha stage of the development process.

Although the current tool used as internal storage (SQLite) for the implemented system does not support concurrency for writing data into the database, migrating the current internal storage to advanced database engines such as MySQL and PostgreSQL, that supports high concurrency is very easy. This was achieved by using the ORM python library, Peewee, which models the database schema. In addition, the database inventory can be easily updated by simply executing the database module of the system. In other words, suggestions provided for specific vulnerabilities can be easily enhanced and modified, the inventory of IoT specific vulnerabilities SeeSec detects can be updated,

and known vulnerable operating system version(s) and encryption algorithms can be updated to ensure the security service provided by SeeSec is up to date.

The implemented framework adds an overhead to the initial installation period, but only on the order of minutes, so it is reasonable for a consumer who is not frequently installing new IoT devices. Once the devices have been checked and whitelisted, there is minimal impact on performance in our system, and the impact could be reduced further if the system is implemented on a traditional networking device.

As future work, user experience enhancements that will make it easier for inexperienced users to understand what vulnerabilities exist and how they will be fixed can be explored. In addition, implementing suitable ways information can be provided to users will be beneficial for the implemented system, considering the feedback from users that participated in the research experiment. Our findings show majority of users prefer email and mobile application as a platform for interacting with the system.

Furthermore, users expressed the desire to have the SeeSec program embedded in a hardware device that will require only a one-time cost to adopt the security service provided. Hence, implementing the software program in an embedded device (especially network devices) is research worthy.

Additional functionality that runs scheduled vulnerability scans on hosts (IoT devices) to make sure the security checks are up to date can be implemented. This functionality can be used to address security issues that occurs after initial installation of an IoT device, and also enforce updated security policies as new vulnerabilities in IoT devices emerge.

The best set of scans to use as vulnerability scanners for IoT devices can be explored. In our current system we are using an extensive vulnerability checker, and three integrated custom scanners. Optimizing these scanners and implementing addition optimal scanners that could reduce the scan while still discovering most IoT specific vulnerabilities will be a significant improvement of the current system. Another worthy aspect for research is building a custom knowledge base for IoT devices to make fixing vulnerabilities and providing information to users more comprehensive and helpful.

BIBLIOGRAPHY

[1] F. Hu and J. N., "Security and Privacy in Internet of Things (IoTs) : Models, Algorithms, and Implementations," 2016.

[2] R. Mahmoud, T. Yousuf, F. Aloul, and I. Zualkernan, "Internet of things (IoT) security: Current status, challenges and prospective measures," in *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, Dec. 2015, pp. 336–341.

[3] M. Ishino, Y. Koizumi, and T. Hasegawa, "A Study on a Routing-Based Mobility Management Architecture for IoT Devices," in *2014 IEEE 22nd International Conference on Network Protocols*, Oct. 2014, pp. 498–500.

[4] K. S. Sahoo, B. Sahoo, and A. Panda, "A secured SDN framework for IoT," in *2015 International Conference on Man and Machine Interfacing (MAMI)*, Dec. 2015, pp. 1–4.

[5] F. Olivier, G. Carlos, and N. Florent, "New security architecture for iot network," in *International Workshop on Big Data and Data Mining Challenges on IoT and Pervasive Systems*, May 2015.

[6] S. Gallagher, "Double-dip Internet-of-Things botnet attack felt across the Internet | Ars Technica," Accessed October 21 2016. [Online]. Available: http://arstechnica.com/security/2016/10/double-dip-internet-of-things-botnet-attack-felt-across-the-internet/

[7] "Security testing the internet of things iot." [Online]. Available: http://www.beyondsecurity.com/security-testing-iot-internet-of-things.html

[8] M. Abomhara and G. M. Kien., "Systems security engineering: Considerations for a multidisciplinary approach in the engineering of trustworthy secure systems," in *NIST Special Publication*, Nov. 2016, pp. 800–160.

[9] I. Andrea, C. Chrysostomou, and G. Hadjichristofi, "Internet of Things: Security vulnerabilities and challenges," in *2015 IEEE Symposium on Computers and Communication (ISCC)*, Jul. 2015, pp. 180–187.

[10] OWASP, "Open web application security project, [online]," May 2017.

[11] C. L. and R. J., "Overview of Security and Privacy Issues in the Internet of Things," in *Security and Privacy Issues in the Internet of Things*, April 2014.

[12] A. Meaola, "How the internet of things will affect security privacy," in *http://www.businessinsider.com/internet-of-things-security-privacy-2016-8?IR=T*, December 2016.

[13] H. O. Jacob Cox Jr., Russel Clark, "Leveraging sdn for arp security," in *IEEE International Conference*, May 2016.

[14] S. Shin and G. Gu, "Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks, (or: How to provide security monitoring as a service in clouds?), in network protocols (lcnp)," in *20th IEEE International Conference on*, November 2012, pp. 1–6.

[15] A. Saeed, A. Ahmadinia, A. Javed, and H. Larijani, "Intelligent intrusion detection in low-power iots," in *ACM Trans. Internet Technol. http://dx.doi.org/10.1145/2990499*, December 2016.

[16] A. Patil, G. Bansod, and N. Pisharoty, "Hybrid lightweight and robust encryption design for security in IoT," in *nternational Journal of Security and Its Applications, http://dx.doi.org/10.14257/ijsia.2015.9.12.10*, September 9 2015, pp. 85–98.

[17] M. Xin, "A Mixed Encryption Algorithm Used in Internet of Things Security Transmission System," in *2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, Sep. 2015, pp. 62–65.

[18] P. Flood and M. Schukat, "Peer to peer authentication for small embedded systems: A zero-knowledge-based approach to security for the Internet of Things," in *2014 10th International Conference on Digital Technologies (DT)*, Jul. 2014, pp. 68–72.

[19] Q. Wen, X. Dong, and R. Zhang, "Application of dynamic variable cipher security certificate in internet of things," in *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*, vol. 03, Oct. 2012, pp. 1062–1066.

[20] P. M. Sanchez, R. M. Lopez, and A. F. G. Skarmeta, "PANATIKI: A network access control implementation based on PANA for IoT devices," in *First workshop on Hot topics in software defined networks, ACM*, November 2013.

[21] S. Barbar, A. Stango, N. Prasad, and R. Prasad, "Proposed embedded security framework for internet of things (IoT)," in *2nd IEEE International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace and Electronic Systems Technology (Wireless VITAE)*, February 2011.

[22] B. Anggorojati, P. N. Mahalle, N. R. Prasad, and R. Prasad, "Capability-based access control delegation model on the federated IoT network," in *2012 15th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, Sep. 2012, pp. 604–608.

[23] M. Leo, F. Battisti, M. Carli, and A. Neri, "A federated architecture approach for Internet of Things security," in *Euro Med Telco Conference (EMTC), 2014*, Nov. 2014.

[24] M. W. Condry and C. B. Nelson, "Using Smart Edge IoT Devices for Safer, Rapid Response With Industry IoT Control Operations," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 938–946, May 2016.

[25] "Beyond trust, retina IoT (RIoT) vulnerability scanner, [online]," Accessed: May 2017.

[26] "Bullguard vulnerability scanner, [online]," Accessed: May 2017.

[27] J. Matherly, "Shodan exploits, 2015 [online]," Accessed: April 2017.

[28] J. Markowsky and G. Markowsky, "Scanning for vulnerable devices in the internet of things," in *The 8th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, September 2015.

[29] C. A. Shue, L. M. Lamb, and N. R. Paul, "United States Patent: 9081960 - Architecture for removable media USB-ARM," Patent 9 081 960, July, 2015, 00000.

[30] K. Zhao and L. Ge, "A Survey on the Internet of Things Security," in *2013 9th International Conference on Computational Intelligence and Security (CIS)*, Dec. 2013, pp. 663–667.

[31] G. L. d. Santos, V. T. Guimares, G. d. C. Rodrigues, L. Z. Granville, and L. M. R. Tarouco, "A DTLS-based security architecture for the Internet of Things," in *2015 IEEE Symposium on Computers and Communication (ISCC)*, jul 2015, pp. 809–815.

[32] "What is dhcp?, [online]," Accessed: May 2017.

[33] B. Dolph, "Making systems more user-friendly, [online]," May 2012.

[34] W. H, H. H, and G.-J. A., "LPM: Layered Policy Management for Software-Defined Networks," in *Atluri V., Pernul G. (eds) Data and Applications Security and Privacy XXVIII. DBSec 2014. Lecture Notes in Computer Science, vol 8566. Springer, Berlin, Heidelberg*, July 2014.

[35] O. M. E. Committee and al., "Software-defined networking: The new norm for networks." in *ONF White Paper. Palo Alto, US: Open Networking Foundation*, April 2012.

[36] N. B, S. M, de Oliveira B, M. C, O. K, and T. T., "Software defined networking enabled capacity sharing in user-centric network," in *IEEE Communications Magazine*.

[37] W. Stallings, "Software-defined networks and openflow - , volume 16, no. 1," in *The Internet Protocol Journal*, March 2013.

[38] S. K. Tayyaba, a. W. N. Naila Khan, M. Shah, Y. Asim, and M. Kamran, *Software-Defined Networks (SDNs) and Internet of Things (IoTs): A Qualitative Prediction for 2020*, November 2016.

[39] L. Elbaz and H. Bar-El, "Strength assessment of encryption algorithms," October 2000.

[40] E. Brown, "Open source operating systems for IoT [online]."

[41] D. Guinard, "Operating systems for IoT embedded systems [online]," December 2016.

[42] "Mitre, CVE (common vulnerabilities and exposures) [online]," Accessed: April 2017.

[43] M. Stanislav and T. Beardsley, "HACKING IoT: A case study on baby monitor exposures and vulnerabilities," pp. 1–17, September 2015.

[44] "https://betanews.com/2016/10/13/hacking-internet-of-things-devices-old-vulnerability/ [online]," in *Accessed: May 2017*.

[45] "https://dyn.com/ [online]," in *Accessed: May 2017*.

[46] "https://krebsonsecurity.com/ [online]," in *Accessed: May 2017*.

[47] "http://mininet.org/ [online]," in *Accessed: May 2017*.

[48] "https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html [online]," 2017.

Appendix A

SOURCE CODES

The source codes of the implemented framework, SeeSec can be accessed on the link provided below.

- Source Code.

  `https://github.com/SecurityInEmergingEnvironments/SeeSec`