Scalable IoT Network Testbed with Hybrid Device Emulation

by

Zhengan Zhao
B.Sc., Miami University, USA, 2020

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Supervisory Committee

Dr. Kui Wu, Supervisor
(Department of Computer Science)

Dr. Jianping Pan, Departmental Member
(Department of Computer Science)

ABSTRACT

In recent years, the Internet of Things (IoT) has been proliferating in various fields, such as health care, smart city, and connected autonomous vehicles. Accompanying the popularity of IoT are security attacks that exploit the vulnerabilities of many IoT devices. To build IoT anomaly detection systems, we need to collect and label a large amount data from diverse IoT scenarios, which is a time-consuming and prohibitive task if without the support of an IoT testbed. This thesis fills this urgent need by developing a scalable, flexible, safe, and secure IoT testbed.

To make the testbed scalable, we need to reduce the hardware cost and make its architecture easily extendable. For this, we build a hybrid testbed consisting of real IoT devices, such as motion sensors and smart cameras, and emulated devices with Raspberry Pi. The emulated devices can replace real IoT devices with the same operational behaviour as real IoT devices but at a much lower price. Flexibility means the testbed can easily simulate different application scenarios. To make the testbed flexible, we build a dedicated data management module to facilitate the complex tasks in generating diverse traffic patterns, reproducing security attacks, collecting, visualizing, and analyzing network traffic. Testbed safety means the testbed will not cause any adverse impact to the Internet, and testbed security means protecting it from outside attacks. For safety, we carefully analyze the source code and the behaviour of launched attacks and configure a traffic filter to strictly contain the attack traffic within the testbed. For security, we scan and analyze the security of all IoT devices within the testbed to ensure no exposed vulnerability in the used devices.

Contents

Supervisory Committee **ii**

Abstract **iii**

Contents **iv**

List of Tables **vi**

List of Figures **vii**

Acknowledgements **x**

Dedication **xi**

1 Introduction **1**

1.1 What Is the IoT System? 1

1.2 IoT Security And Vulnerabilities 4

1.3 Motivation 7

1.4 Background on Testbed 8

1.5 Contributions and Thesis Organization 10

2 An Analysis on IoT Attacks **12**

2.1 The Purpose of Analyzing IoT Attacks 12

2.2 Introduction of Typical Malware Attack Mechanisms 13

2.2.1 Vulnerabilities of IoT Software and Firmware 13

2.2.2 Malware 17

3 Emulating IoT Device with Raspberry Pi **21**

3.1 The Necessity of Building Emulated IoT Devices 21

3.2 A Brief Introduction of the Raspberry Pi Platform 23

3.2.1 Raspberry Pi Hardware 23
3.2.2 Raspberry Pi OS 26
3.2.3 Raspberry Pi Add-on Devices 26
3.3 Emulating IoT Devices with Raspberry Pi 27
3.4 Encrypting IoT Data 33
3.5 Conclusion 36

4 Testbed Architecture and Establishment 37
4.1 Overall Architecture of Testbed 37
4.2 Testbed Configuration 40
4.2.1 IoT Devices in the Testbed 40
4.2.2 Traffic Mirroring 42
4.2.3 Security and Safety Measures 46
4.2.4 Diverse Communication 51

5 Data Collection and Analysis 54
5.1 Motivation for Collecting IoT Network Traffic Data 54
5.2 Overview of Collected Data 55
5.2.1 Status 55
5.2.2 Scenarios 56
5.3 Statistical Analysis 57
5.4 Device Security Analysis 67

6 Conclusion and Future Work 71
6.1 Conclusion 71
6.2 Future Work 72

Bibliography 74

List of Tables

Table 1.1 Operating system layered architecture 5
Table 1.2 OSI network layered architecture 5

Table 2.1 Vulnerable devices for CVE-2019-10999. 14

Table 3.1 Raspberry Pi 3 A+ Tech Specs 24
Table 3.2 Raspberry Pi 3 B Tech Specs 25
Table 3.3 Raspberry Pi 4 B Tech Specs 25
Table 3.4 Raspberry Pi Zero W Tech Specs 25

Table 4.1 List of IoT Devices. 41
Table 4.2 Information of the host machine. 47
Table 4.3 Information of the virtual machine. 47

Table 5.1 Captured data summary 55
Table 5.2 Data collection by scenarios 56
Table 5.3 Major vulnerabilities on devices. 69
Table 5.4 Devices with major vulnerabilities. 69

List of Figures

Figure 1.1 IoT and non-IoT active device connections worldwide from 2010 to 2025. The years with * means the expectation years. The data is from [24, 53]. 2
Figure 1.2 Classification of IoT application areas. 4
Figure 1.3 The layered architecture for IoT 5

Figure 2.1 The back-end of target device after exploitation. 16
Figure 2.2 A conceptual diagram of Mirai botnet. 18

Figure 3.1 Different Raspberry Pi models. 24
(a) Raspberry Pi 3 A+ 24
(b) Raspberry Pi 3 B 24
(c) Raspberry Pi 4 B 24
(d) Raspberry Pi Zero W 24
Figure 3.2 Desktop version of Raspberry Pi OS. 26
Figure 3.3 Raspberry Pi add on devices. 27
(a) Raspberry Pi Camera Module 27
(b) Sense HAT 27
Figure 3.4 Emulated webcam with Raspberry Pi 4B and Raspberry Pi lens. 28
Figure 3.5 Client-side view of live video streaming. 30
Figure 3.6 Emulated IoT server-client architecture. 31
Figure 3.7 Emulated temperature sensor: the server side. 32
Figure 3.8 Emulated temperature sensor: the client side. 32
Figure 3.9 Emulated temperature sensor: sample data 33
Figure 3.10TLS files. 34
(a) TLS certificate authority file 34
(b) TLS private key file 34
Figure 3.11Sample TLS data traffic. 34
Figure 3.12Sample plain text. 35

Figure 3.13TLS-encrypted data for the plain text in Fig. 3.12. 35

Figure 4.1 The diagram of lab testbed. 37
Figure 4.2 OpenWrt interface of router GL-AR750S-EXT. 43
Figure 4.3 Commands to set *iptables* rules. 43
Figure 4.4 Sample pcap file showing the success of traffic mirroring. 44
Figure 4.5 Interface of router's GUI system for port mirroring. 45
Figure 4.6 Interface of the virtual machine. 47
Figure 4.7 Modified part in malicious code. 48
(a) Scanner header file. 48
(b) Initialization of the scanner. 48
Figure 4.8 Command to open firewall file. 49
Figure 4.9 An example of new firewall rule. 49
Figure 4.10Restart the firewall. 49
Figure 4.11Ping external network. 50
Figure 4.12Ping within testbed. 51
Figure 4.13WPS switch on webcam. 52
Figure 4.14WPS switch on router. 53

Figure 5.1 Webcam working with live video on, number of packets in time interval of 1 second. 58
Figure 5.2 Webcam working with live video on, data size (Bytes) in time interval of 1 second. 59
Figure 5.3 Webcam data collected in the lab, number of packets in time interval of 1 second. 60
Figure 5.4 Webcam data collected in the lab, data size (Bytes) in time interval of 1 second. 60
Figure 5.5 Webcam data collected in a residential location, number of packets in time interval 1 of second. 61
Figure 5.6 Webcam collected in a residential location, data size (Bytes) in time interval of 1 second. 61
Figure 5.7 Webcam live video in the presence of network congestion. . . . 62
Figure 5.8 Traffic during vulnerability exploitation. 63
Figure 5.9 Payload of CVE-2019-10999. 64
Figure 5.10The Mirai loading process stands out w.r.t. the number of packets per second. The webcam is in the live video mode. 65

Figure 5.11The Mirai loading process stands out w.r.t. with data size (Bytes) per second. The webcam is in the live video mode. 66
Figure 5.12A Mirai-controlled webcam lunches DoS attack. 67
Figure 5.13Sample Nessus scan GUI. 68
(a) Devices and vulnerability status. 68
(b) Single vulnerability description. 68
Figure 5.14The weight of discovered vulnerabilities. 70

ACKNOWLEDGEMENTS

I would like to thank:

Supervisor, Dr. Kui Wu, for mentoring, support, guidance, encouragement, and patience.

Committee members, Dr. Jianping Pan and Dr. Hong-chuan Yang, for serving in my defence committee and giving insightful comments.

Yang Zhou, Benny Huang, and Ronnie Salvador Giagone for their helpful discussion and feedback.

Huawei Technologies Canada Co. Ltd. for the funding support of this research.

My family for supporting me in the low moments.

Thank you.
Zhengan Zhao

DEDICATION

To my family

Chapter 1

Introduction

1.1 What Is the IoT System?

In recent years, the Internet of things (IoT) is becoming a widely-used technology in industries and people's daily lives. The idea of IoT was proposed as early as the mid-1970s, even though the term of IoT had not been used yet. In the 1970s, a Coke vending machine at Carnegie Mellon University was connected to the Advanced Research Projects Agency Network (ARPANET), the Internet's predecessor, to solve practical problems, such as remotely checking the temperature inside the vending machine [62]. This attempt already had all the essential parts for an IoT device, such as the sensors, network connection, processing capability, and identifier. And it also met the primary purpose of IoT devices, which is offering services to various applications by interconnecting things in physical and virtual approaches [61]. In 1999, the term "Internet of things" was created by Kevin Ashton, the executive director of the Auto-IDCentre, MIT [4]. In the past 20 years, IoT devices have experienced explosive growth. In the meantime, the IoT supporting technologies have become mature. For example, Radio Frequency Identification (RFID), IEEE 802.11 (Wi-Fi), Zigbee, and Bluetooth have been widely adopted to support IoT communications [55].

While the global consumption of electronic products has continued to grow in recent years, the growth rate of IoT devices is remarkably faster than other types of devices. From multiple sources, global IoT device usage was far less than non-IoT devices in the 2000s, and in the late 2010s, the number of the two was almost equal. In the next few years, however, the number of IoT device deployments will far exceed

the number of non-IoT devices [7, 19, 30, 37, 52]. As shown in Figure 1.1, there will be three times as many IoT contrivances as non-IoT devices in 2025. Such rapid growth of IoT devices is due to the booming IoT applications in the following major fields:

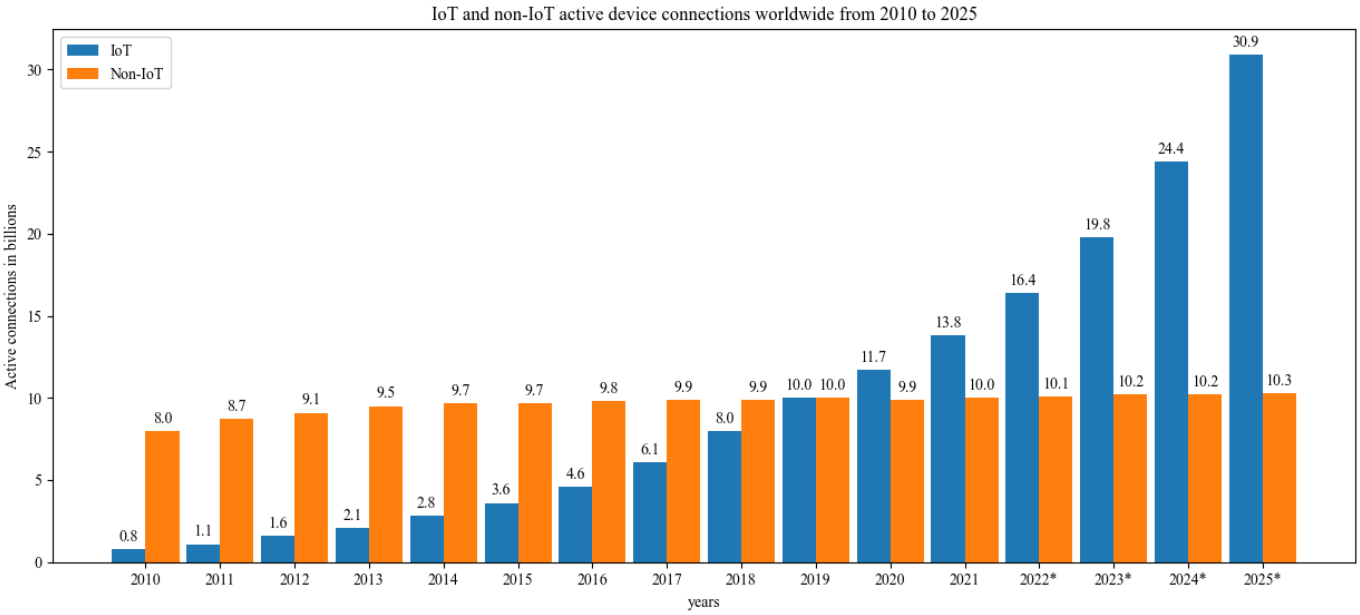


Figure 1.1: IoT and non-IoT active device connections worldwide from 2010 to 2025. The years with * means the expectation years. The data is from [24, 53].

- **Medical applications:** The Internet of Medical Things (IoMT) is used for vital signs monitoring, elder care, telemedicine, and medical data collection. These applications can improve the efficiency of hospital operations and help healthcare workers make diagnoses [12, 16, 20, 33]. As the technology of IoMT matures, more professional applications, such as Remote patient monitoring (RPM) devices and Personal emergency response systems (PERS), have been developed. While IoMT brings convenience, it also creates problems and challenges. Since the IoMT devices usually have direct contact with the human body, they must ensure not causing harm to humans. If the IoMT devices collect patients' information, they must ensure privacy and prevent personal information leakage. The reliability of the devices is also necessary, since the IoMT devices may need to collect accurate health data in different scenarios (e.g., walking, jogging, and running).

- **Infrastructure applications:** With the increasing scale of cities, many new challenges have arisen for urban management. The concept of Smart City was

proposed for deal with the challenges [23], by supporting intelligent services for energy management, garbage recycling, water treatment, public safety, etc [54]. The goal of the smart city is to efficiently manage the city, provide high-quality services to citizens, and reduce labor and energy consumption by deploying a large number of IoT devices. This concept has been applied in many large cities, such as Singapore, Dubai, Oslo, Copenhagen, and Boston [32]. Nevertheless, the smart city is facing challenges in cyber security [63]. Attacks on infrastructure have occurred frequently in recent years. The most significant impact is the cyber attack on the North American oil pipeline in 2021, which caused energy shortage and inconvenience to people's lives [15, 59].

- **Industrial applications:** The application of IoT in the industrial field is also called IIoT. Two typical example IIoT are manufacturing and agriculture. In the manufacturing field, IoT is applied to smart factories, robotic arm control, supply chain management, and energy consumption control. Due to their small size, portability, and low price, IoT devices can be embedded in various production machinery so that users can observe and control the production process or automate the process. As such, IIoT can effectively reduce business costs and improve production efficiency. Some notable examples are Industry 4.0 and Tesla Gigafactory [17, 21, 23, 50, 56, 65]. In agriculture, IoT devices can monitor the natural environment, such as weather, soil, pests, and plant growth, based on which farmers can automate seeding, irrigation, and fertilization. These technologies can maximize the utilization of agricultural resources and reduce risks and costs for users [1,13,28,40]. For IIoT, reliability and confidentiality are about business productivity and business confidentiality. Nevertheless, attacks against IIoT are common [58].

- **Consumer applications:** The smart home and wearable IoT devices are integral parts of consumer applications. IoT devices have been used for power management, voice control, multimedia streaming, and house cleaning, providing people with the most immediate convenience and entertainment in their daily lives. The functions and computing power of a single smart home or wearable device are relatively weak, but today's smart home devices can be connected to a mobile phone or computer through a gateway to form a network of IoT devices interacting within a home or office [29]. Some typical smart home products include Amazon Alexa, Apple Home Kit, and Google Nest. It is worth

noting that among several IoT applications, the threats faced by smart homes and wearable devices are more severe due to the lack of strong security protection in most home IoT devices. Once compromised, these IoT devices may disclose consumers' personal information to the Internet. Even worse, the compromised IoT devices can be exploited for distributed deny of service (DDoS) attacks to cause more significant damages [5, 22, 48].

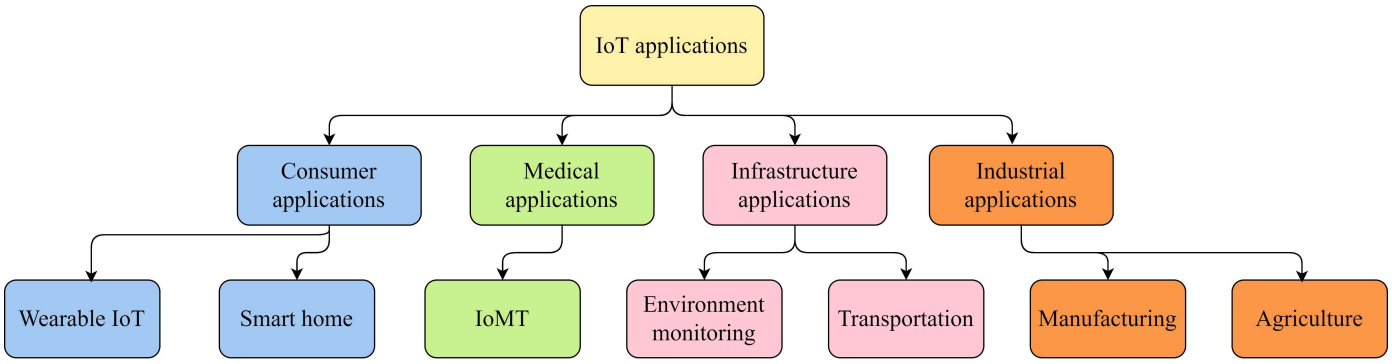


Figure 1.2: Classification of IoT application areas.

Figure 1.2 shows a brief classification of IoT applications. Among various types of IoT applications, we mainly focus on the consumer applications in this thesis. Consumer IoT devices, such as smart cameras and smart doorbells, are closely related to people's daily life but usually do not include strong security protection. According to research by Hewlett-Packard (HP), many consumer IoT devices vulnerabilities have been exposed, such as insufficient authorization, insecure interfaces, or lack of transport encryption. It has been reported that 60% to 80% IoT consumer products were vulnerable [14, 46]. In the next section and chapter 2, we will discuss the vulnerabilities of consumer IoT devices and attacking mechanism in more detail.

1.2 IoT Security And Vulnerabilities

To find out IoT devices' vulnerabilities, we should first understand their architectures. As an emerging concept, IoT has special architecture. For better understanding, we start with well-established architectures for operating systems and computer networks. For the former, Dijkstra developed the layered architecture for operating systems as shown in Table 1.1 [9]. For the latter, a typical example is the Open Sys-

tems Interconnection model (OSI model), which divides the data flow in the computer networks into seven layers, as shown in Table 1.1.

Table 1.1: Operating system layered architecture

Layer 6	User programs
Layer 5	I/O Management
Layer 4	Operator Console
Layer 3	Memory Management
Layer 2	CPU Scheduling
Layer 1	Hardware

Table 1.2: OSI network layered architecture

Layer 7	Application
Layer 6	Presentation
Layer 5	Session
Layer 4	Transport
Layer 3	Network
Layer 2	Data link
Layer 1	Physical

The architecture of IoT devices does not follow strictly the conventional OS model or network model, but rather have particular characteristics depending on specific application scenarios. While currently there is no consensus or standard for IoT architecture, the 3-layer general model is widely accepted, including the perception, network, and application layers [25, 31, 39, 47, 64, 66], as shown in Figure 1.3.

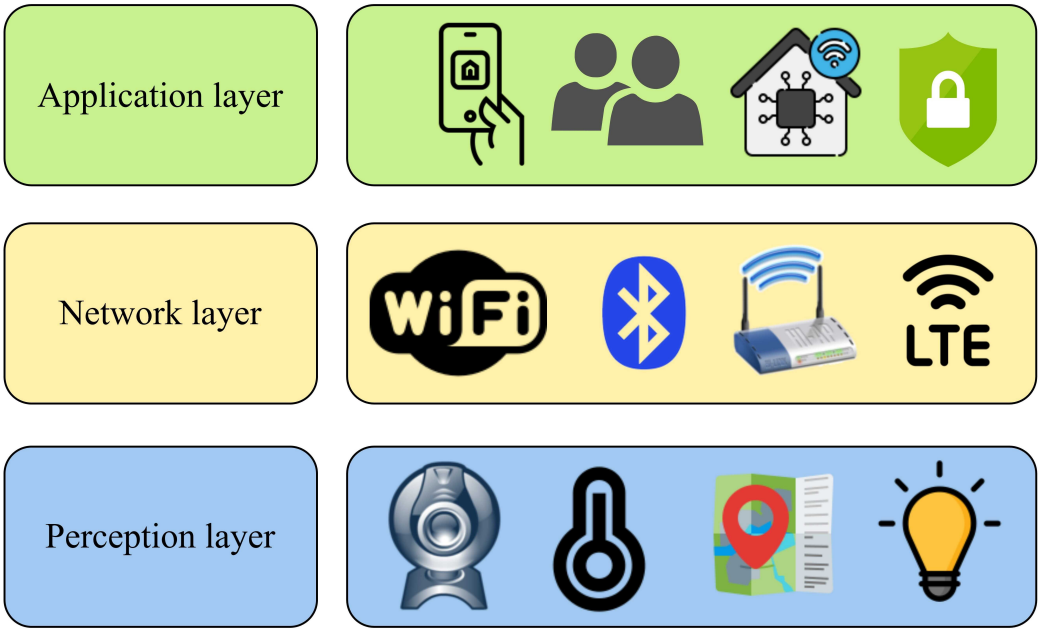


Figure 1.3: The layered architecture for IoT

This thesis follows the 5-layer IoT model when discussing IoT vulnerabilities. Next, we introduce the functions of each layer and their vulnerabilities [2, 8, 25, 26, 38, 41, 60, 66].

- **Perception layer:** The perception layer is also called the physical layer or sensor layer, and it is the foundation of IoT. Its main components may include QR codes, barcodes, RFID, cameras, and various sensors. The main functions of the IoT perception layer are information perception and raw data collection. In this layer, the vulnerabilities are tightly coupled with specific physical hardware. For instance, RFID has been widely used in access cards, passports, and other IoT devices for identification and tracking. However, there is the risk of illegally modifying, tracking, and blocking RFID.

- **Network layer:** The network layer collects data from the perception layer and provides the function of data routing, transmission over the Internet. The network gateways play an important roll, they are the mediation between different networks or IoT devices. Various communication technologies can be used in this layer, e.g., Wi-Fi, Long-Term Evolution (LTE), Bluetooth, near-field communication (NFC), and Zigbee. Some vulnerabilities have been reported against these technologies, for example, the Bluejacking and Bluebugging [6,36] attacks against Bluetooth.

- **Application layer:** The application layer collects data from the network layer to provide various services to the user. In this layer, various application protocols are applied, such as Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), HyperText Transfer Protocol (HTTP), and Telnet. There are some critical vulnerabilities in this layer. The Mirai botnet can intrude on an IoT device by default Telnet and SSH service.

In some of the more-refined categories, there is a business layer on top of the application layer. The business layer provides the business-related services such as authority management, device firmware update, data analysis, and cloud computing. Usually, the services at this layer are maintained by the application service provider (ASP), Internet service provider (ISP), Original Equipment Manufacturer(OEM), or cloud providers. It is also worth noting that different IoT applications may have quite different architecture. For example, industrial IoT may have a massive perception layer, but it may not need an application layer and the data is transmitted from the network layer to the business layer directly [60]. The diversification of network data and the increase in the number of users open the door for security attacks aiming at the business layer. For example, some attackers can forge a large number of fake

users for data poisoning and inject a massive amount of incorrect data to decrease the accuracy and efficiency of the cloud server.

Overall, we can see that each layer of IoT architecture is subject to secure attacks, which are prevalent due to the several reasons. First, the IoT devices' hardware and software lack standardization [2], and many proprietary protocols may have undetected flaws. Second, the CPU architectures in the IoT device market are diverse, including ARM, MIPS, and their numerous variants. The hardware complexity creates the complexity of the operating system, applications, and remote services, making a unified security solution nearly impossible. Finally, many IoT devices use weak passwords or insecure APIs, and in many cases a simple brute-force dictionary attack may compromise the IoT devices. We will discuss some typical IoT attacking mechanisms in more detail in Chapter 2.

1.3 Motivation

In order to improve the security of IoT devices and defend against network attacks, researchers must strengthen the security of IoT device software and develop effective network anomaly detection systems for IoT. This is premised on understanding the different mechanisms of cyber-attacks and preparing appropriate datasets for testing anomaly detection algorithms and training machine learning models.

To achieve the above goals, need to build a testbed that can help us reproduce and analyze the attack process in the IoT settings. In this way, the researchers can understand the mechanism of the attack and propose more effective defence. At the same time, this testbed also needs to collect diverse data that are representative for IoT applications, such as data generated when IoT devices work regularly and data generated when an attack occurs. The datasets will be used for our research in IoT anomaly detection, e.g., federated learning-based anomaly detection and root cause analysis.

Building a realistic IoT testbed requires non-trivial engineering efforts and needs to carefully balance the tradeoff between the scale and the cost. Most real-world IoT applications involve hundreds or even thousands of IoT devices. Ideally, the testbed should reflect the similar IoT environment with a large number of devices of different device types. Nevertheless, replicating large-scale IoT systems over the testbed is prohibitive in both hardware and data management costs. Due to this reason, building a large-scale IoT test is normally out of the reach of academic research

labs. To address this problem, we are motivated to build *an IoT testbed that is cheap, easily extendable, and collects diverse IoT data useful for building/testing IoT anomaly detection systems.*

1.4 Background on Testbed

Based on the establishment method, there are three types of testbed often used in research:

- **Real-device testbed:** All evaluations, testing, and data collection are based on commercially available IoT products. All data and behaviors are generated by replicating real-world IoT applications. This type of testbed is most expensive, but the experimental results and the data from the testbed are most close to real-world IoT systems.

- **Simulated testbed (aka simulator):** A simulator is a software system that behaves like a real-world system to perform network tests and experiments. This type of testbed is normally termed as simulator to distinguish from real-device testbed, in the sense that the implementation of the simulator is totally different from the real-world object's implementation. For example, Network Simulator 3 (NS3) is well-known network simulation software, and it simulates a real-world network on one computer by writing scripts in C++ or Python. NS3 can create virtual nodes, links, or packets. A simulator offers researchers the convenience and freedom to create a virtual network with different topology and configurations. However, there may be a certain gap between the authenticity of its data and the data generated in the real-device testbed. It is also hard for a simulator to test real-time human-machine interaction or the interaction between IoT sensor and physical environment, further limiting its value in building/testing IoT anomaly detection systems.

- **Emulated testbed:** An emulator is a hardware or a software system that behaves like a real-world object to perform network tests and experiments. Unlike simulators, the implementation of the emulator is the same as the real-world object's implementation. For example, a smart camera emulator uses the same approach as a commercial smart camera to interact with the environment, e.g., video-recording the environment and streaming the video to the cloud.

Nevertheless, if we use Raspberry Pi to emulate a smart camera, we can simply generate pre-recorded videos from Raspberry Pi without using any physical lens. In this way, we can greatly save cost (a Raspberry Pi is much cheaper than a smart camera) without sacrificing much on the authenticity of data. Another advantage of emulated testbed is that it provides a degree of freedom for controlling and modifying the emulated devices. For instance, we can use Raspberry Pi to emulate a temperature sensor, a light sensor, or a smart camera.

Based on the testbed usage, there are three types of testbeds often used in research as below:

- **Security-oriented testbed:** Such a testbed is used for security research. For safety reasons, the testbed is usually not connected to the Internet or uses a firewall to strictly contain the traffic, because some real cyberattacks are reproduced on the testbed. The testbed may use actual or emulated IoT consumer products. There is also a control center for researchers to test attacks and security scans of IoT devices. The output of such a testbed is usually a security assessment report for multiple IoT devices or a small IoT network. A typical such test platform is "SoK," a security evaluation testbed for home-based IoT applications [3], which hosts 45 IoT devices such as webcams, home assistants, media, and network devices. The authors [3] tested the IoT devices with multiple tools, such as Nessus Network Monitor, ntopng, and Wireshark, and used *sslsplit* to generate a man-in-the-middle attack. Based on the tests, they produced a security report, including each IoT device's security ratings, features, and vulnerabilities.

- **Performance-oriented testbed:** This type of testbed has a similar architecture to a security-oriented testbed, but for a different purpose. Usually, it is used to test the accuracy and efficiency of a detection system or a program. When testing a new protocol or new software, it can measure CPU usage, memory usage, network latency, and power consumption. Researchers have established such testbed to test the time efficiency, and CPU usage for an IoT architecture used in smart cities and agriculture [67]. Another such testbed is built to evaluate the performance of IoT middleware for syntactical interoperability [45], i.e., the packaging and transmission mechanisms for data communication between different protocols. Their focus is on evaluating the CPU/memory utilization and transmission delay [45].

- **Real-time monitoring and data collection testbed:** This type of testbed may mix security-oriented and performance-oriented testbed functionalities. Its purpose is to monitor the devices' behavior in real-time and collect devices' behavior data for security and/or performance analysis. For example, when the researchers analyze the cyber-attack pattern, they need to build such testbed to ensure they produce plenty of data. For this purpose, a physical IoT testbed is established to collect network traffic and use the data for botnet analysis [34]. To be effective, such testbed should include a dedicated component for collecting, managing, and analyzing large-scale network traffic data.

Based on the above introduction, we conclude that the emulated testbed is cheaper than real-device testbed, and it generates more realistic data than the simulator. With the emulated testbed, we can achieve the goals stated in the previous section. This thesis follows the emulated approach but uses both real and emulated IoT devices in the testbed. In addition, our testbed combines the functionalities of a security-oriented testbed and data collection testbed.

1.5 Contributions and Thesis Organization

This thesis makes contributions by tackling the technical challenges in building a safe, scalable, and flexible IoT testbed. In this thesis, we use the term "*safety*" to refer to that the testbed will not cause adverse impacts to the Internet and the term "*security*" to mean that the testbed is protected from outside attacks. We mainly focus on the safety issues while relying on existing secure measures for security.

- **Safety:** Because the main purpose of the testbed is for security analysis on IoT devices and anomaly detection systems, some malware will be launched to attack the IoT devices. For an IoT device such as smart camera to work normally, the IoT device may need to communicate to cloud services controlled by the product's service provider. In other words, the testbed needs to connect to the Internet to test the normal behavior of IoT devices. When the malware is launched, we need to make sure no attack traffic can escape the testbed to cause any damage to the public. To address this challenge, we carefully analyze the behavior of any launched attacks and configure a traffic filter to strictly contain the attack traffic within the testbed. (Chapters 2 and 4)

- **Scalability:** A scalable IoT testbed should use low-cost hardware for cost saving, and its architecture should be easily extendable. To address this challenge, we build a hybrid testbed consisting of both real IoT devices, such as motion sensors and smart cameras, and emulated devices with Raspberry Pi. The emulated devices can be used to replace real IoT devices with same operational behavior as real IoT devices and without much sacrifice on the authenticity of network traffic. (Chapters 3 and 4)

- **Flexibility:** One of the testbed's purposes is to collect network traffic under different IoT scenarios for further research. As such, the testbed must generate the network traffic for different scenarios, including heavy/light background traffic, different maximum transmission unit (MTU) settings, and different stage of malware attacks (e.g., scanning, launching, heartbeating, attacking). Flexibility also implies ease of data management for collecting, classifying, and reproducing network traffic for different test scenarios. To achieve this goal, we build a dedicated data management module to facilitate complex tasks in data collection, visualization, and analysis. (Chapter 5)

The rest of the thesis is organized as follows. Chapter 2 consists of the motivation to analyze the attack mechanism of typical IoT devices. A concrete examples is given to show how open-source malware can be used to reproduce the attack on our testbed. The knowledge presented in this chapter will be utilized to ensure the safety of the testbed when we design the testbed architecture in Chapter 4. Chapter 3 includes the particular purpose of building emulated IoT devices using Raspberry Pi. We introduce the Raspberry Pi platform, and the details of emulated IoT devices. Chapter 4 presents the overall architecture and the detailed setup of the testbed. We also explain why the architecture is easily extendable. Chapters 3 and 4 together address the scalability issue of the testbed. Chapter 5 introduces the dedicated data management module, including data collection, data generation for different scenarios, IoT network traffic analysis, and devices security analysis. Finally, we conclude the thesis and discuss future work in Chapter 6.

Chapter 2

An Analysis on IoT Attacks

2.1 The Purpose of Analyzing IoT Attacks

There are many different types of IoT attacks, and it is impossible to analyze all of them. Even so, we need to gain a basic understanding of the general attack strategies for two main reasons: First, we need to launch malware attacks in the testbed to test IoT devices' security measures. Second, we need to isolate the attack traffic inside the testbed. Without a good knowledge of the attack mechanism, it would be hard to contain the attack traffic. Third, we need to study the traffic patterns during different stages of attacks, which requires us to know the behaviour of attacks. Overall, we need to clarify the following issues:

- How to implant the malware into an IoT device?
- What is the structure and component of the malicious code?
- How to deploy and configure malicious code and its control center?
- How to launch the attack on specific IoT devices?

In the rest of this chapter, we start with a general introduction on typical malware attack mechanism and then use Mirai botnet as the concrete example to illustrate the attack details.

2.2 Introduction of Typical Malware Attack Mechanisms

2.2.1 Vulnerabilities of IoT Software and Firmware

As described in Chapter 1, there is weakness in various aspects of IoT that attackers can exploit. The lack of manufacturing standards and low-cost hardware components are possible reasons for this phenomenon.

General speaking, the attack against IoT can be divided into two main steps. The first step is to discover and exploit some vulnerabilities in the IoT device itself. Compared to attacking computer servers, this task is relatively easier since IoT device manufacturers usually do not implement strong security protection for cost saving. At the end of the first step, attackers can gain the access to control the device. The second step is to implant malware into the IoT device through exposure. In this section, we will mainly discuss the first step.

Many organizations or websites publish the newly discovered IoT vulnerabilities, for example, the Common Vulnerabilities and Exposures (CVE) and the National Vulnerability Database (NVD) include abundant information regarding IoT vulnerabilities. According to 2021 statistics from CVE [27], the weaknesses that are exploited more often against IoT devices include:

- **Buffer overflow**: A buffer overflow occurs when a buffer receives more data than it can store. Buffer overflow data usually corrupts program and results in unexpected termination. In addition, if the contents of the overflow data are intentionally constructed, it is feasible to execute the specific command or take control of the system. Such vulnerabilities can be caused by various reasons, from poor programming to problems with the programming language itself. For example, assembly and C/C++ allow direct memory access, granting enough power for attackers to develop buffer overflow attacks.

- **SQL injection**: An SQL injection attack inserts a malicious SQL query or adds a statement into a program's input parameters. The SQL query is then executed on the backend SQL server. In order to completely hijack an IOT device, the attacker needs to assume root level of control of it. One of the easiest ways to achieve this is by using an SQL injection.

- **Directory traversal attack**: Directory traversal is a security vulnerability caused by insufficient security validation of user-entered file names by a web server or web application. The implementation does not adequately filter user input for "../", allowing an attacker to bypass the server's security restrictions and access arbitrary files by using some special characters.

In this thesis, for a detailed description of vulnerabilities exploitation, we use CVE-2019-10999 as an example [10]. Nowadays, many programs can analyze firmware, such as Centrifuge, a firmware analysis platform from ReFirm Labs. With the tool, people have discovered Vulnerabilities, such as CVE-2019-10999, against several models of D-link webcams. CVE-2019-10999 is a stack-based buffer overflow vulnerability in *alphapd*, the back-end server program for D-link cameras that allows an authenticated user to execute arbitrary code by passing a long string to the WEPEncryption parameter when requesting webpage *wireless.htm*. It is a common vulnerability for the following devices:

Table 2.1: Vulnerable devices for CVE-2019-10999.

Model	Firmware version
DCS-5009L	1.08.11 and below
DCS-5010L	1.14.09 and below
DCS-5020L	1.15.12 and below
DCS-5025L	1.03.07 and below
DCS-5030L	1.04.10 and below
DCS-930L	2.16.01 and below
DCS-931L	1.14.11 and below
DCS-932L	2.17.01 and below
DCS-933L	1.14.11 and below
DCS-934L	1.05.04 and below

Below, we use the D-link DCS932L model as an example to demonstrate how to exploit the device's weaknesses to gain control of the device. We use open-source code from GitHub to implement the exploitation [18]. The open-source exploitation code needs to download on a Linux machine. The code needs four parameters to run: the target device's IP address, target port number, target device's username, and password. We write a shell script, *hack.sh*, where the open-source exploitation code is *exploit.py*, the target device's IP address is 192.168.0.12, the port number is 80, the username for the back-end system is "admin", and the password for the back-end

system is set to “5212352123”.

python3 exploit.py -i 192.168.0.12 -P 80 -u admin -p ”521235123”

Then we use the command *./hack.sh* to execute the code. After the code is successfully executed, we could gain the administration control of the back-end system, and we can execute any command in the back-end of target device, as shown in Figure 2.1.

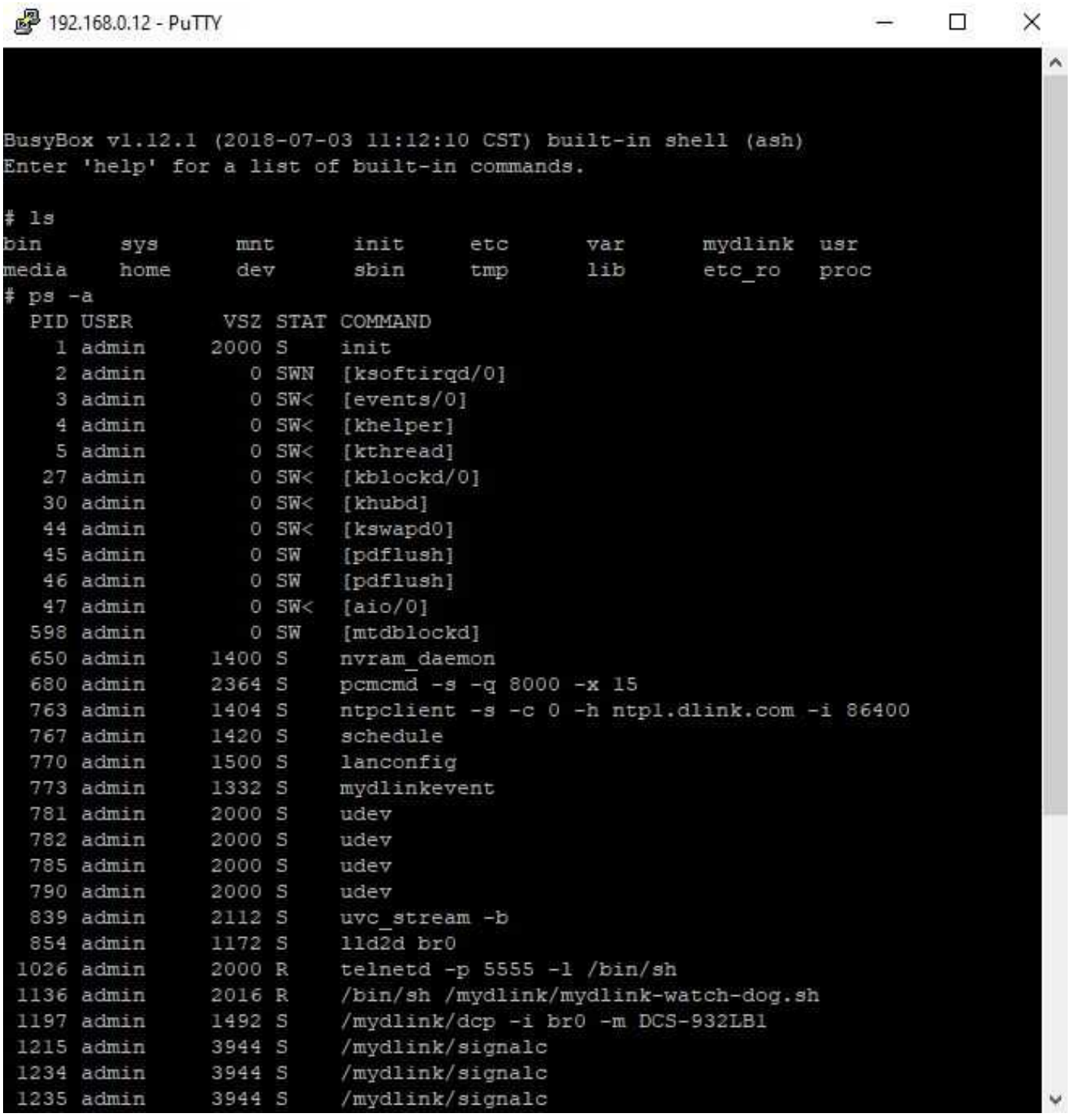


Figure 2.1: The back-end of target device after exploitation.

2.2.2 Malware

Once certain ports or services of IoT devices are exploited, attackers can implant malwares into these target devices. In Chapter 1, we introduced various attack methods and malwares against IoT devices. Botnet is most dangerous, and Mirai is one of the most representative botnet malware due to its broad impact on a large number of IoT devices. The Mirai botnet was first found in a large range of distributed denial of service attacks (DDoS). It has been reported that Mirai attacks can cause victim IoT devices loss of control, disconnect victim hosts from the Internet, or even crash victim websites. Mirai can launch various types of attacks, such as Valve Source Engine flood, DNS resolver flood, SYN flood, AC flood, TCP stomp flood, and HTTP flood. The domain flux technique could be used in botnet malware to make the network security administrator hard to detect, block, and back trace the attack. It is achieved by dynamically generating domain names with domain generation algorithm (DGA). The earliest Mirai botnet event was found in 2016 [11]. Soon afterwards, there appear a lot of it's variants. As botnet malwares, Mirai and it's variants likely use the domain flux technique. While this technique is usually used by real-world Mirai botnet, it is not relevant to our testbed environment.

In this section, we analyze the behavior of Mirai and explain how to deploy and use Mirai to generate attack traffic on our testbed. The reason that we analyze the nuts and bolts of Mirai malware is twofold: (1) understand the structure of the most impactful malware in IoT attacks, and (2) develop a testbed for IoT intrusion detection, over which we can launch attacks that have the similar structure as Mirai and capture both benign and attack traffic for training and evaluation detection models.

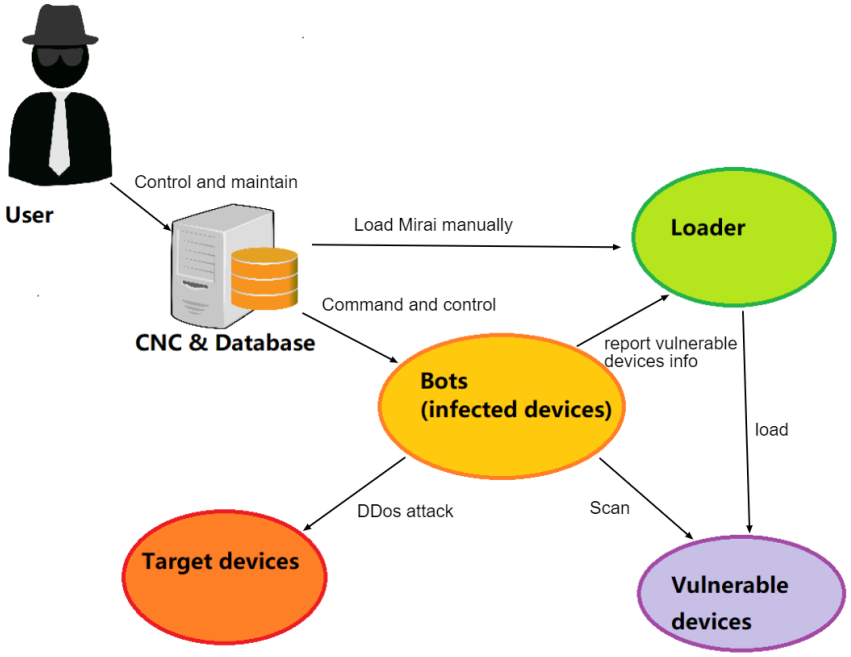


Figure 2.2: A conceptual diagram of Mirai botnet.

The conceptual diagram of Mirai botnet is shown in Figure 2.2. Generally speaking, Mirai botnet consists of cross compilers, command-and-control (CNC) server, scanner, loader, and MySQL database. The CNC server is the central controller of the Mirai malware. Based on Mirai source code and related documentation, we summarize the procedure of launching/propagating Mirai attacks as follows:

- **Step 1: (Preparing the attack)** Mirai first needs to scan the Internet to identify members for inclusion in the botnet. For this purpose, Mirai maintains a report server and randomly probes systems on the Internet. Note that in a lab testbed, this step can be skipped.

- **Step 2: (Deploying the malware and building the botnet)** The malware code is cross-compiled for a variety of architectures, such as *miri.X86*, *mirai.MIPS*, and *mirai.SH4*. Once a vulnerable victim device is discovered, the loader will identify the architecture of the victim device and load the corresponding executable. For example, *mirai.X86* could be used on most PC, because most PC use X86 CPU, and *mirai.MIPS* could be used for a large family of IoT devices using MIPS CPU. With the executable running, the victim device is now a member of the botnet and extends the attack following the same scanning and attack activities as any other node in the botnet.

The bots (i.e., infected IoT devices in Step 2) communicate with the CNC server to receive attack commands and report connection status. The attack commands may include different types of attacks. For example, the command "http" means HTTP flood attack, and "?" means print the available attack list. The SQL database is established locally with the CNC server. The SQL database includes mainly three tables to store users' information, attack history, and a whitelist. The scanner is used by the mirai botnet to scan for more vulnerable IoT devices over the Internet. And the loader is used to load the binary executable into the victim devices. Once a victim is identified, the loader first checks if there is *wget* (a program that retrieves content from web servers) or *tftp* (Trivial File Transfer Protocol) available in the target device. If yes, the malware will be loaded on the device with *wget* or *tftp* directly. Otherwise, the loader first uses the *echo* command to download a small binary into the target device, and this binary will serve as the function of *wget*. Once the executable successfully runs on the victim device, the victim device will be connected to the CNC, which means a new bot has been created. The botnet is built and is ready for launching DDoS once a (large) number of bots have been created.

- **Step 3: (Launching attack with DDoS)** After the botnet is built, the attacker can launch a DDoS attack with command line. A simplest attack command could be like this:

  [*bot number*] [*attack type*] [*targets*] [*duration time*]

  The [*bot number*] is the number of bots to attack at the same time. This number depends on the total number of bots in the botnet. The [*attack type*] could be "http", "udp", or "tcp". The [*targets*] denote the IP addresses of the victims, and there could be up to 255 targets to attack in a time. The [*duration time*] is an integer value between 1 and 3600 in the unit of second. For example, an attack command:

  1 udp 192.168.0.1 120

  means using one bot to perform a UDP flood attack on the ip address *192.168.0.1* for 120 seconds.

- **Step 4: (Cleaning the "household")** This step is not relevant to our testbed but is usually included in real-world Mirai botnet to improve its operation. Mirai source code includes mechanisms to cover bot tracks and block competitors. To keep itself from discovery, Mirai deletes itself from the file system once the malware is running, and it changes its name to a randomized value. To protect itself from competing botnets, Mirai looks for certain identifiers associated with competing botnets. If it finds any (the competitors are hardcoded in the source code), it will kill that process.

After we introduce the mechanism of the malicious software targeting IoT devices, we can have an overall idea of establishing the testbed. It is worthy mentioning that we study the Mirai source code for research purpose only. For safety and security, we strictly contain the Mirai botnet in the lab environment and will block any attack traffic outgoing our local testbed.

After studying the source code and the runtime behavior of Mirai, we draw the following insights:

- The mechanisms behind the malware are complex, and most mechanisms are specific to IoT vulnerabilities.
- If malware is not carefully contained within the testbed, there is a high risk that the attack traffic will leak out.
- All data in the attack process are identifiable based on the knowledge gained from the above analysis, and the detection and collection of these data will be of great help in improving the security of IoT devices.
- According to the malware's operation mechanism, we need to build the testbed that facilitates malware tests and, in the meanwhile, safeguards attack traffic. For example, we need to set up a network firewall to block attack traffic outgoing to the Internet. We also need a central server to collect all the data generated in the testbed.

Chapter 3

Emulating IoT Device with Raspberry Pi

3.1 The Necessity of Building Emulated IoT Devices

Single-board computers (SBCs) have been widely used for IoT software and hardware development. For instance, Raspberry Pi has been used by many researchers for the development of various IoT devices and application prototypes, such as IoT-based biometrics implementations [51], air quality monitoring systems [35], and smart home automation techniques [44]. In this thesis, we use Raspberry Pi to build emulated IoT devices. The main idea of building emulated devices is to keep the main implementation approach of the target imitating object (e.g., commercial IoT devices) using much cheaper hardware and simplified functionality. The functional simplification is to trim down the full-stack function of a commercial IoT device to partial function that our evaluation is focused on. For instance, if our goal is to develop testbed for data collection and further research on IoT network anomaly detection systems, we may exclude some functions, such as device registration to cloud center, from the emulated devices.

While we have discussed the main benefit of using emulated devices in Chapter 1, we further justify the rationale of developing emulated IoT devices for the testbed as below:

- The capability of controlling the data in the transmitted network packets is

necessary for anomaly detection system research. For example, when testing the performance of a context/payload-based IoT network anomaly detection system, we may need to create some data streams artificially. The content of these data packets needs to cover plain text, audio, video, and other forms. However, commercial IoT products may not meet this need because we have no control over the packets transmitted by these devices. For example, most of these devices encrypt data whose original plain text may not be available to researchers. If we want to compare the system performance with and without encrypted data, we have to generate plain data with the emulated IoT device and then encrypt the data in the same conditions.

- The primary use of this testbed is to generate and collect data during malware attacks. However, not all IoT devices have known vulnerabilities that can be exploited for successful attacks. Even if a vulnerability is discovered and published on a site like CVE, the open-source exploitation code is not available in most cases. In addition, most IoT devices do not open up the ports and services of background programs to the users. As a result, we may only have a limited number of devices that we can use for security research, since we cannot implant malware and generate attack data from the compromised devices. Using emulated devices, we can effectively avoid this problem for we have full control on device.

- One of the critical features of IoT devices is their interaction with the real world. In order to generate data that closely resembles the real world, we want to follow the basic IoT architecture mentioned in Chapter 1 for performance evaluation. Therefore, we need a realistic perception layer that contains a variety of sensors, such as cameras, temperature sensors, and humidity sensors. Emulated devices with Raspberry Pi can easily meet this requirement.

- IoT devices emulated with Raspberry Pi have many other advantages, including more flexible functionality and a much lower cost. A Raspberry Pi can be loaded with a motion sensor, humidity detector, temperature detector, microphone, or camera. Furthermore, any algorithm or program can be easily deployed on the Raspberry Pi. Moreover, an emulated device is much cheaper than commercial products. For instance, a commercial smart camera easily costs over 200 CAD, but the emulated camera costs less than 75 CAD (Raspberry Pi typically costs

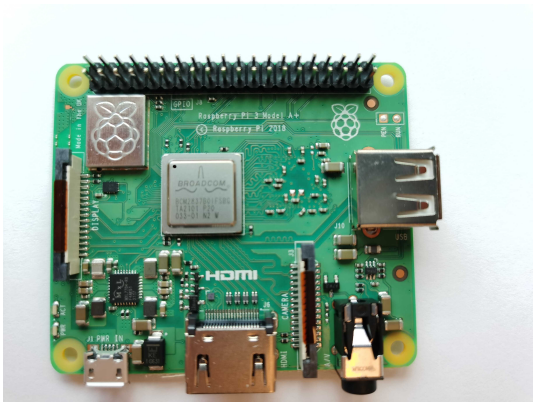
less than 50 CAD and Lens Board for Raspberry Pi Camera costs less than 25 CAD).

3.2 A Brief Introduction of the Raspberry Pi Platform

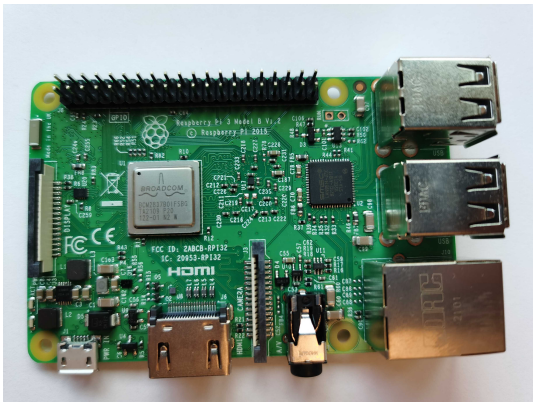
The Raspberry Pi is the most popular single-board computer today. There are similar products such as ASUS Tinker Board, Onion Omega, Banana Pi, and Odroid. However, Raspberry Pi has gained popularity for its extensive community support and excellent price/performance ratio. In the following, we introduce different models of Raspberry Pi devices, Raspberry Pi operating system (OS), and some add-on devices, which we use in our testbed.

3.2.1 Raspberry Pi Hardware

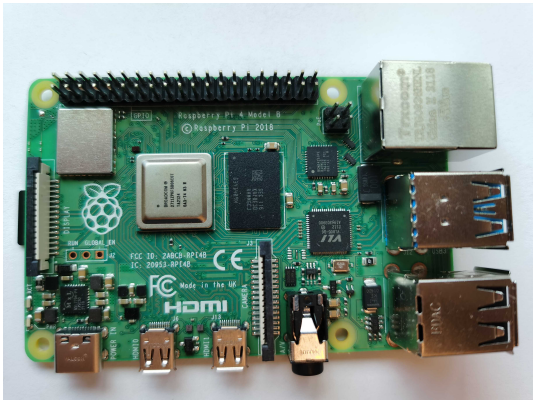
We use Raspberry Pi Zero 2 W, Raspberry Pi 4, Raspberry Pi 3 Model A+, and Raspberry Pi 3 Model B. All the models have wired and wireless network connections and necessary ports to connect the add-on devices. These devices are shown in Figure 3.1, with their specification listed in Table 3.1, Table 3.2, Table 3.2, and Table 3.4.



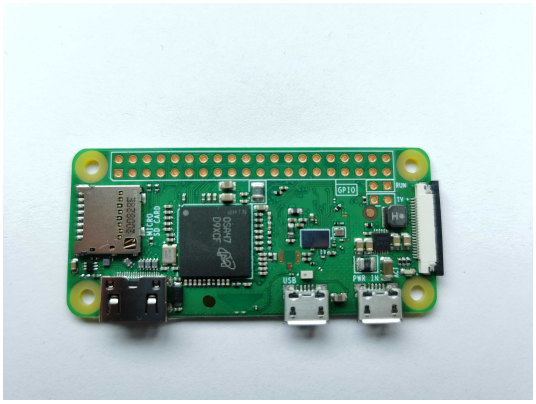
(a) Raspberry Pi 3 A+



(b) Raspberry Pi 3 B



(c) Raspberry Pi 4 B



(d) Raspberry Pi Zero W

Figure 3.1: Different Raspberry Pi models.

Table 3.1: Raspberry Pi 3 A+ Tech Specs

CPU	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
RAM	512MB LPDDR2 SDRAM
Network	2.4GHz and 5GHz 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2/BLE
Ports	Extended 40-pin GPIO header Full-size HDMI Single USB 2.0 ports CSI camera port for connecting a Raspberry Pi Camera Module DSI display port for connecting a Raspberry Pi Touch Display 4-pole stereo output and composite video port Micro-SD card port

Table 3.2: Raspberry Pi 3 B Tech Specs

CPU	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
RAM	1GB LPDDR2 SDRAM
Network	2.4GHz & 5GHz 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, Gigabit Ethernet
Ports	Extended 40-pin GPIO header Full-size HDMI 4 USB 2.0 ports CSI camera port for connecting a Raspberry Pi Camera Module DSI display port for connecting a Raspberry Pi Touch Display 4-pole stereo output and composite video port Micro-SD card port

Table 3.3: Raspberry Pi 4 B Tech Specs

CPU	Broadcom BCM2711, Quad core Cortex-A72 64-bit SoC @ 1.5GHz
RAM	8GB LPDDR4-3200 SDRAM
Network	2.4 GHz and 5.0 GHz 802.11ac wireless, Bluetooth 5.0, Gigabit Ethernet
Ports	2 USB 3.0 ports Raspberry Pi standard 40 pin GPIO header 2 × micro-HDMI ports 2-lane MIPI DSI display port 2-lane MIPI CSI camera port 4-pole stereo audio and composite video port Micro-SD card port

Table 3.4: Raspberry Pi Zero W Tech Specs

CPU	Broadcom BCM2835 single-core 1GHz
RAM	512MB LPDDR2 SDRAM
Network	802.11 b/g/n wireless LAN, Bluetooth 4.1, Bluetooth Low Energy (BLE)
Ports	Mini HDMI port and micro USB On-The-Go (OTG) port HAT-compatible 40-pin header Composite video and reset headers CCSI camera connector

3.2.2 Raspberry Pi OS

Variable operating systems can run on Raspberry Pi, such as Ubuntu, OSMC, LibreELEC, or Windows IoT Core. Nevertheless, we use the Raspberry Pi 64-bit OS on the devices because Raspberry Pi OS is particularly optimized for the CPU and the controller on the Raspberry Pi board that uses the ARM architecture. Raspberry OS is essentially a Linux-based operating system. Figure 3.2 shows an example that the desktop version is installed on a Raspberry Pi 4 B model device. For those Raspberry Pi devices that do not support the desktop version, they can be installed with Raspberry Pi OS Lite, which is a Linux-based OS without any graphic user interface (GUI).

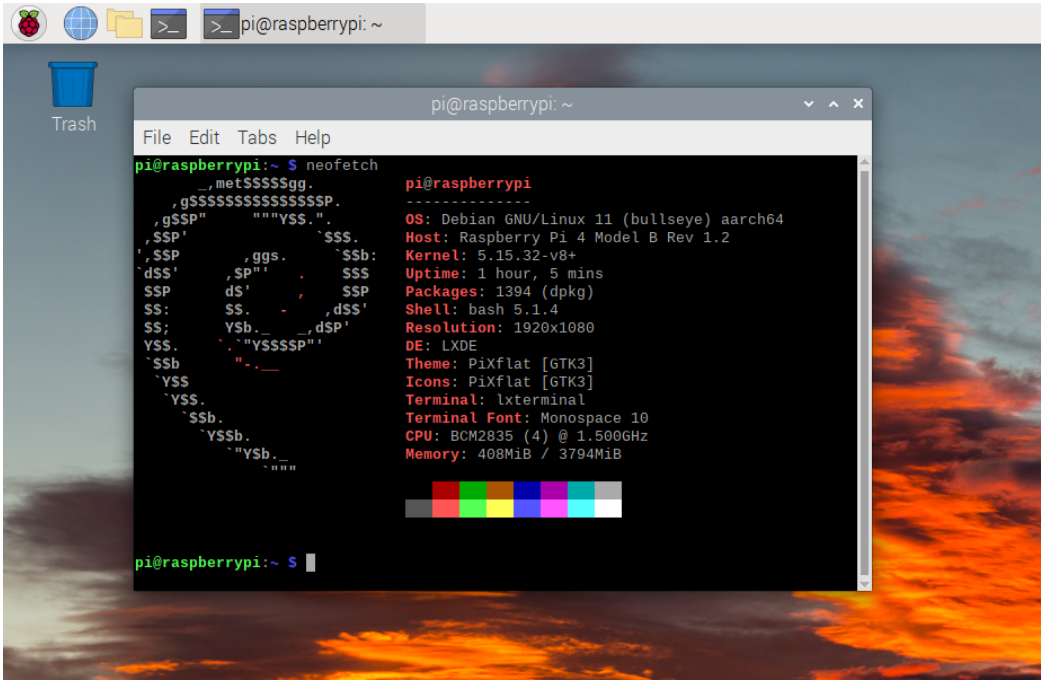
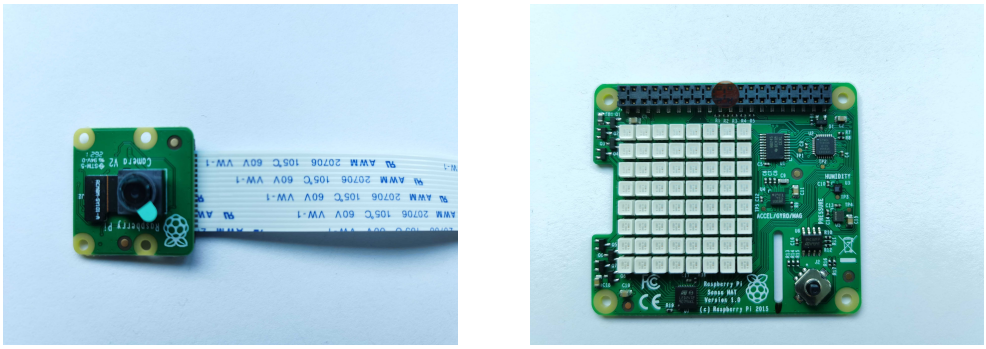


Figure 3.2: Desktop version of Raspberry Pi OS.

3.2.3 Raspberry Pi Add-on Devices

We use various add-on devices to the Raspberry Pi for emulating different types of IoT devices. In this thesis, we use a Raspberry Pi camera module to emulate smart camera, and use Sense HAT, which is combined with an integrated gyroscope, accelerometer, magnetometer, thermometer, barometer, and hygrometer, to emulate different sensing devices. The add-on devices are shown in Figure 3.3.



(a) Raspberry Pi Camera Module (b) Sense HAT

Figure 3.3: Raspberry Pi add on devices.

3.3 Emulating IoT Devices with Raspberry Pi

This section introduces the details of emulating IoT devices with Raspberry Pi. In order to build a useful emulated IoT device that can be used to generate data for IoT anomaly detection research, we also need to build a related control module.

In this thesis, we use the client-server model to emulate the services of IoT devices, such as webcams and thermometers. Below, we first use the emulated webcam as an example to explain the detail, and then explain the same architecture can be used to emulate other types of IoT devices such as thermometers.

We use a Raspberry Pi 4B and a Raspberry Pi Camera Module to establish an emulated webcam. The camera module is attached to the MIPI CSI camera port on the board, as shown in Figure 3.4. Note that the lens can be replaced with other types of cameras, such as USB cameras.

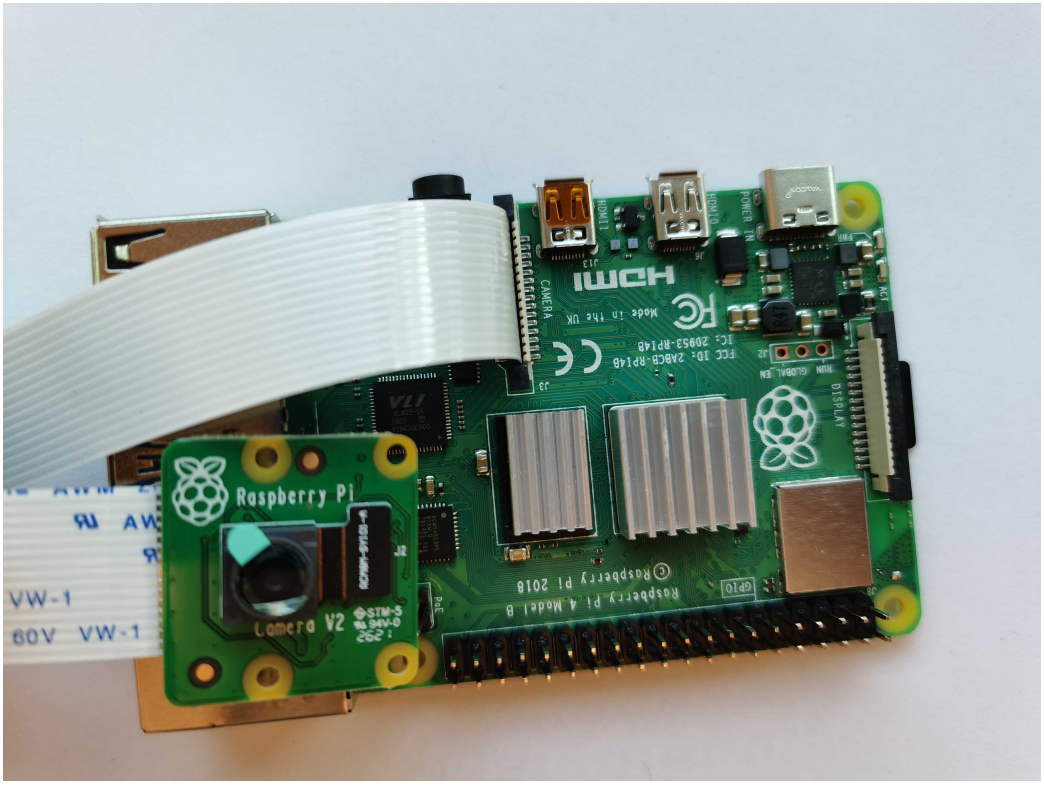


Figure 3.4: Emulated webcam with Raspberry Pi 4B and Raspberry Pi lens.

Then, we use the preset functions on Raspberry Pi or write a client-server program on the emulated IoT devices and the client devices. The emulated IoT devices provide the resource and service, so they are on the server side. We then use another Linux machine as the client, which requests services from the server. In the context of emulating the webcam and generating stream data, we use the "motion" library in Raspberry Pi OS. The "motion" allows out-of-the-box video streaming to a web browser. The "motion" library can be installed by:

sudo apt install motion

Once the "motion" library is installed successfully, we can check and modify the video streaming port in /etc/motion/motion.conf file, the configuration file for the motion library. The file includes the critical system settings, for example,

 # The mini-http server listens to this port for requests (default: 0 = disabled) stream_port 8081

which sets the default video stream port as 8081. If we need a different port for this service, we can change this port number and save the modification. Then, we can run the following command line to start the live video stream:

sudo systemctl start motion

Nevertheless, at this moment, the data stream cannot be generated because the client-side has not been established yet. If we use some preset libraries or services, such as "motion", we can open the browser on another machine, which should be on the same LAN as the emulated IoT devices. In the URL address, we should input the IP address of the emulated IoT devices and the port number of the video stream service. In this example, the IP address is 192.168.0.101, and the port number is 8081, as shown in Figure 3.5. With the browser at the client side, we can see that the live video stream is working as expected, indicating that the steaming data is generated successfully. Between the client and the server, we can use another machine as the monitoring machine to capture all the streaming data with Wireshark. In Chapter 4, we will introduce how to set up the monitoring machine. At this moment, we have set up a complete emulated camera working in our testbed. The architecture for data streaming and capture is shown in Figure 3.6.

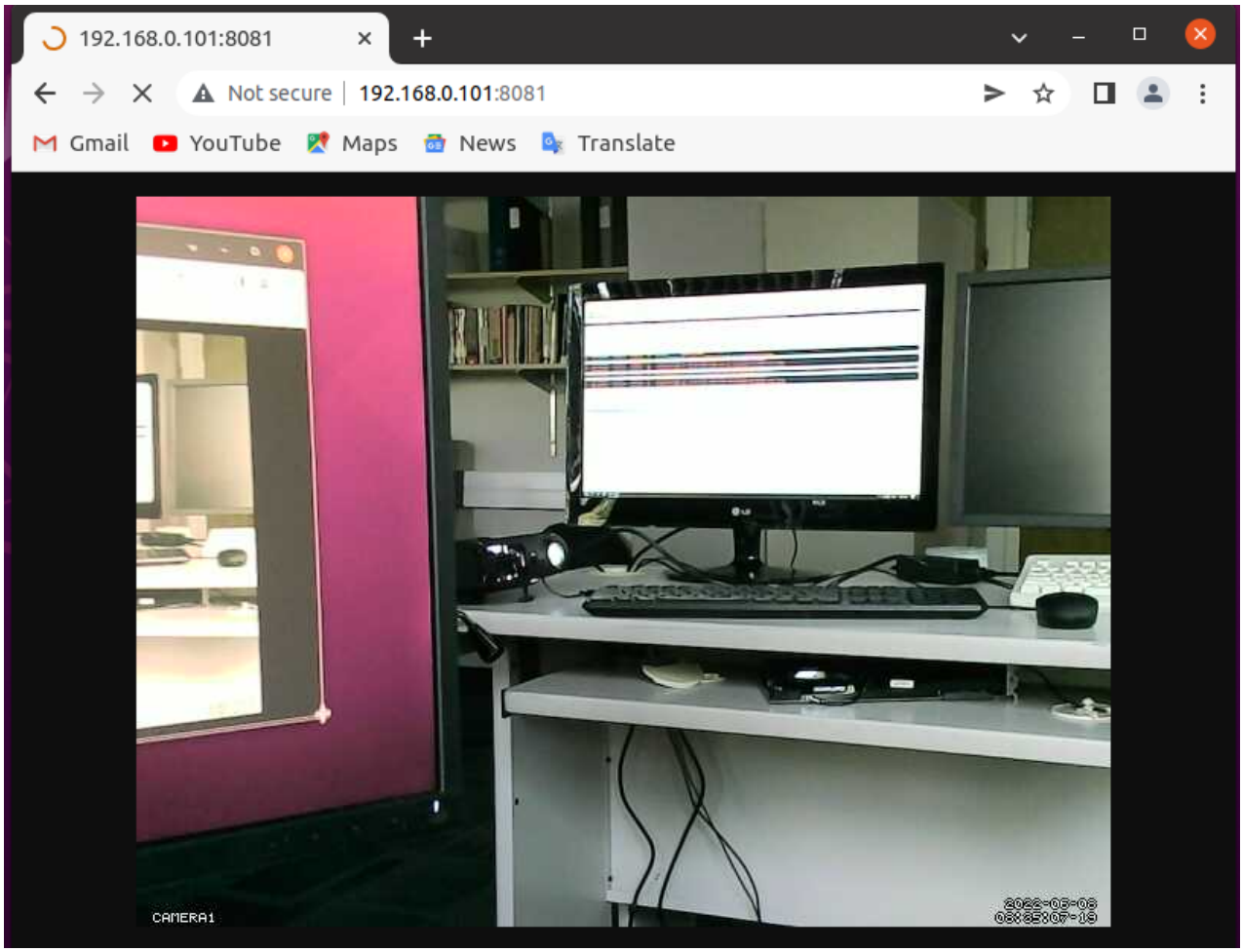


Figure 3.5: Client-side view of live video streaming.

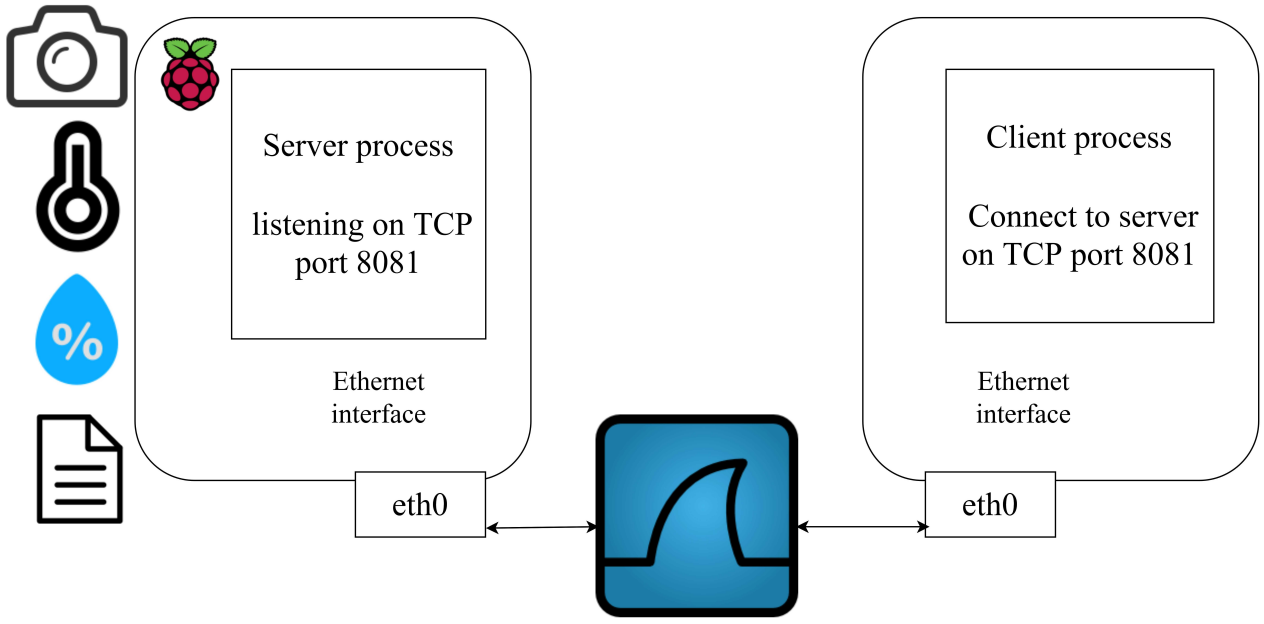


Figure 3.6: Emulated IoT server-client architecture.

It is worth noting that the preset services or libraries are not always available or useful for our testbed setup. In this case, we write our own client-server program. For instance, there are no preset services for emulated temperature & air pressure sensors. We then write a Python program to achieve the same purpose (i.e., streaming and capturing sensor data). In the example shown in Figure 3.7, the IP address of the emulated sensor is 192.168.0.101, and the port number 5432 is used for this service. The client side is shown in Figure 3.8. The captured streaming data is shown in Figure 3.9.

```
from sense_hat import SenseHat
import socket
import time
import datetime
import sys

sense = SenseHat()
sense.clear()
HOST = "192.168.0.101"
PORT = 5432
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()
counter = 0
while True:
    conn, add = s.accept()
    counter = counter + 1
    pressure = sense.get_pressure()
    temp = sense.get_temperature()
    humidity = sense.get_humidity()
    data = str(counter) + ": " + "pressure: " + str(pressure) + ": "\
           + "temp: " + str(temp) + ": " + "humidity: " + str(humidity)\
           + " " + str(datetime.datetime.now())
    conn.send(str(data).encode())
    while True:
        time.sleep(3)
        counter = counter + 1
        pressure = sense.get_pressure()
        temp = sense.get_temperature()
        humidity = sense.get_humidity()
        data = (str(counter) + ": " + "pressure: " + str(pressure) + ": "\
               + "temp: " + str(temp) + ": " + "humidity: " + str(humidity)\
               + " " + str(datetime.datetime.now()))
        conn.send(str(data).encode())
```

Figure 3.7: Emulated temperature sensor: the server side.

```
import socket

HOST = "192.168.0.101"  # The server's hostname or IP address
PORT = 5432  # The port used by the server
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
while True:
    s.send(b"recv")
    data = s.recv(1024)
    print(f"Received {data!r}")
```

Figure 3.8: Emulated temperature sensor: the client side.

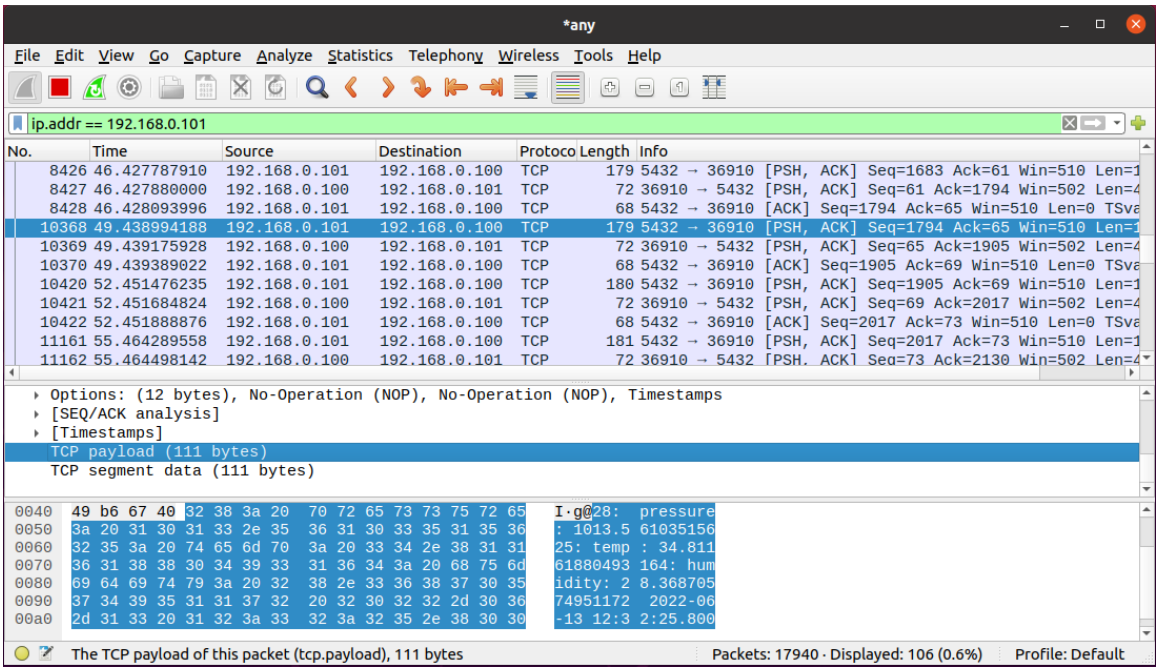


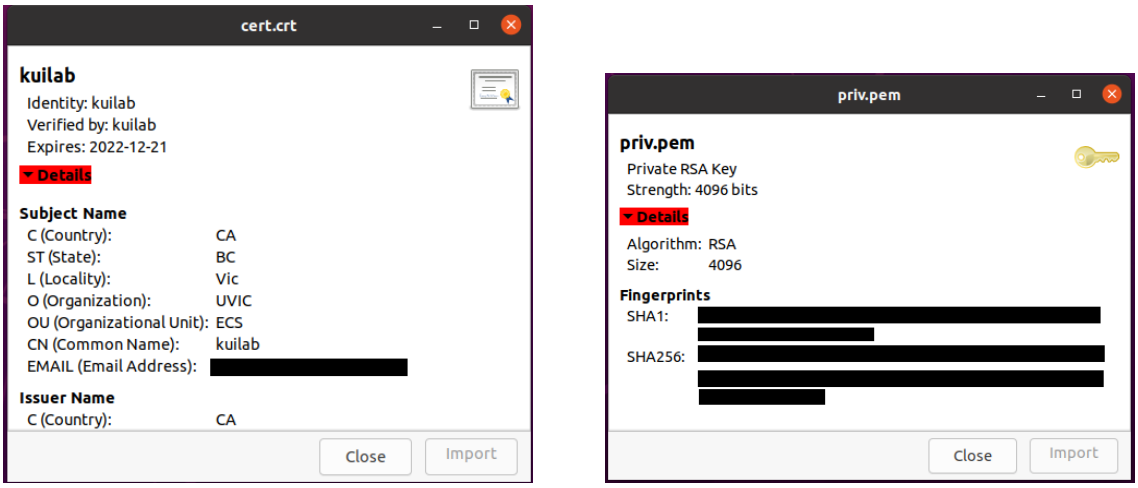
Figure 3.9: Emulated temperature sensor: sample data

3.4 Encrypting IoT Data

So far, we can emulate various IoT devices and services. In the testbed, we can use this architecture to generate various contents, such as video, audio, and plain text. To emulate the behavior of real IoT devices truthfully, however, we also need to support encryption primitives in the emulated IoT devices because most commercial IoT devices encrypt data for security and privacy reasons. In addition, supporting encryption is a much-needed feature of the testbed, because analyzing and classifying encrypted IoT data is a trending research topic.

To implement the data encryption in the testbed, we use the Transport Layer Security (TLS) encryption for the emulated IoT devices when they emulate some IoT functions and services. The basic idea of the TLS protocol is to provide identification and authentication channels for both sides of the communication, thus ensuring the confidentiality and data integrity of the communication. For this purpose, we leverage the TLS implementation in the Python SSL module. After we create our own Certificate Authority file and digital signature file, shown in Figure 3.10, we generate the streaming data with TLS encryption. A sample TLS encrypted data traffic is shown in Figure 3.11. It is easy to observe the difference between encrypted

and non-encrypted plain text data traffic; here, it shows an example of plain text encryption. In Figure 3.12, the payload is plain text, but in Figure 3.13, the payload is unreadable.



(a) TLS certificate authority file

(b) TLS private key file

Figure 3.10: TLS files.

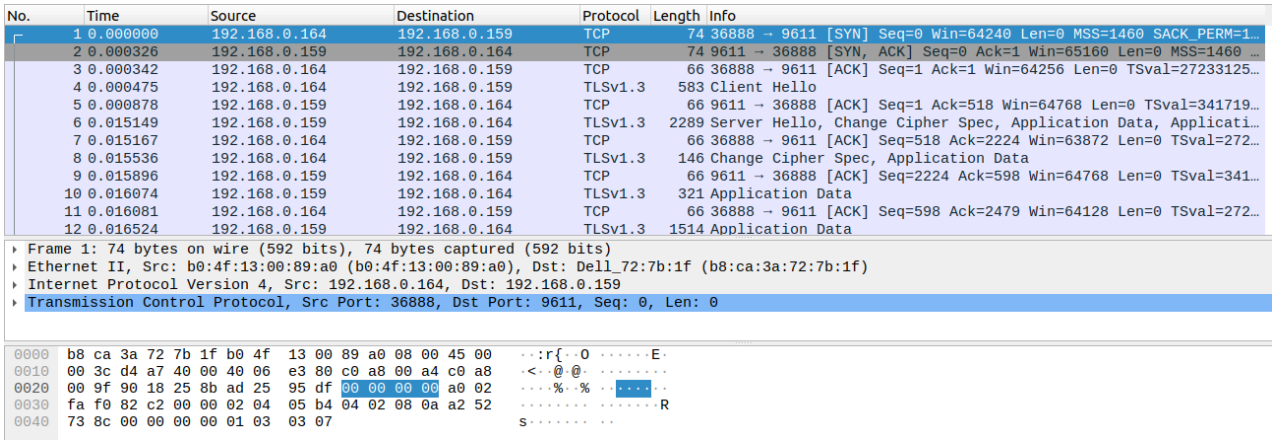


Figure 3.11: Sample TLS data traffic.

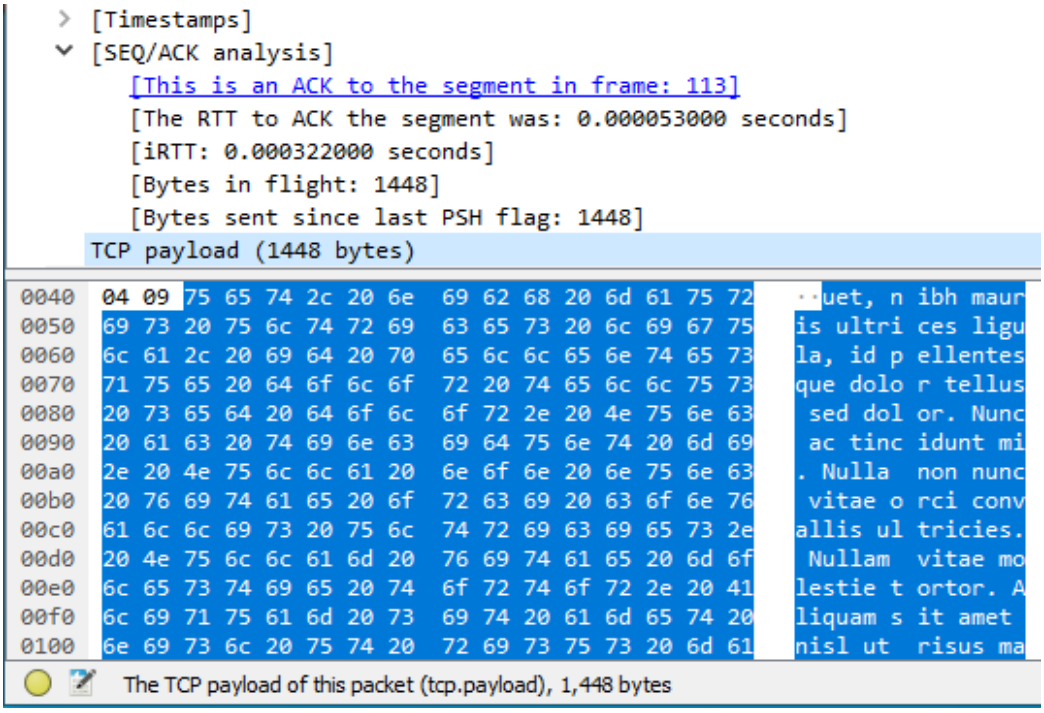


Figure 3.12: Sample plain text.

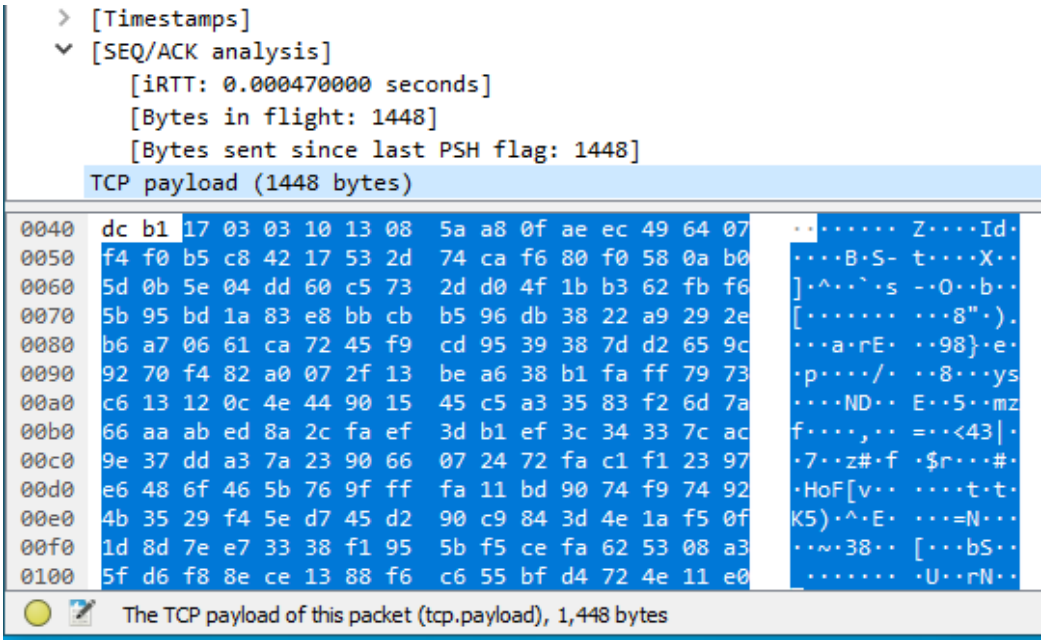


Figure 3.13: TLS-encrypted data for the plain text in Fig. 3.12.

3.5 Conclusion

In this chapter, we introduced the motivation and detailed steps for building emulated IoT devices. We used the Raspberry Pi platform and add-on devices, built client-server program, and captured IoT data from the emulated devices. To summarize, the emulated IoT devices offer the following benefits to our testbed:

- Emulated IoT devices help expand the scale of our testbed, since emulated devices are much cheaper than their commercial counterparts.
- Emulated IoT devices greatly facilitate security-related research. Since the Raspberry Pi OS is Linux-based system and is completely open to users, we can easily access/control the port and services, such as telnet, Secure Shell Protocol (SSH), or File Transfer Protocol (FTP). In addition, we can easily plant malware like Mirai on emulated IoT and control the behavior of the malware (e.g., filtering out attack traffic, blocking victims that use public IP). This task is extremely challenging to perform on commercial products.
- Emulated IoT devices can be used to generate diverse data. For instance, we can configure the emulated device to send out traffic following a pre-defined pattern/model. We can also customize the data for different research purposes, such as developing payload-oriented anomaly detection system.
- The emulated devices offer opportunities for new research. For example, the same application can transmit data with and without TLS encrypting, which helps us better understand the impact of the payload's encryption on the detection model's accuracy.

Chapter 4

Testbed Architecture and Establishment

4.1 Overall Architecture of Testbed

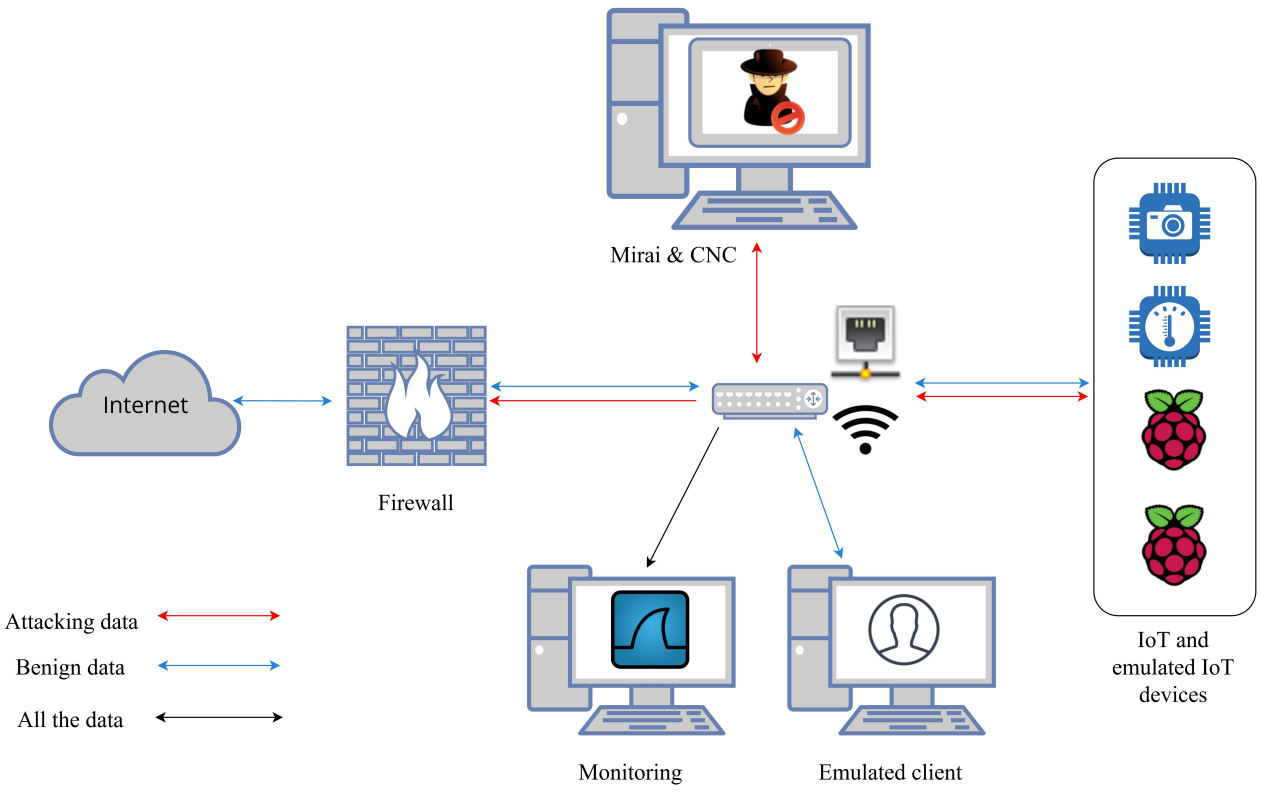


Figure 4.1: The diagram of lab testbed.

The previous three chapters have described IoT development, the features of various test platforms, IoT vulnerabilities, attacks against IoT devices, and the construction of emulated IoT devices. This chapter focuses on building the testbed by integrating the previously mentioned concepts and methods.

The current testbed mainly consists of seven parts as shown in Fig. 4.1.

- One Linux machine loaded with Mirai code and CNC,
- One PC as the monitor,
- Routers,
- Firewall,
- Emulated client,
- Emulated IoT devices,
- IoT devices.

The center router is connected to the Internet through a WAN wallport at UVic. Between the Internet and the internal network of the testbed is a firewall, which is used to guarantee the safety of the testbed, i.e., strictly containing attack traffic inside the lab environment and blocking potential attacks from outside. The firewall is implemented on a router with OpenWRT.

The first PC, which is labeled as "Mirai & CNC" in Fig. 4.1, is connected to the router on one of its LAN ports. On this PC, we launch a virtual machine running Ubuntu, the CNC server, SQL database, and Mirai code. In this thesis, we emulate the CNC in the testbed for safety and flexibility reasons. However, in the real world, the attacker and CNC can be anywhere.

The second PC, which is labeled as "Monitoring" in Fig. 4.1, connected to the router on one of its LAN ports. On this PC, we use Wireshark to capture/monitor all the data flow through the router. The monitoring PC should be connected to the center router on one of its LAN ports. This PC monitor the testbed by using the port mirroring function. We can establish the port mirroring function in two approaches: using command to configure the router with OpenWrt system and using router's GUI for configuring the traffic mirroring function. The details of traffic mirroring will be introduced later in this chapter.

The machine labeled with "Emulated client" works, as the client side, with the emulated IoT devices, as motioned the Chapter 3. This machine's job is generating the client requests to the emulated IoT devices. Then, the emulated IoT devices can generate data traffic of various types, e.g., live video, audio or temperature readings.

The center router, the wireless access point (AP), and other extended routers are working together to ensure the connectivity of the testbed. The AP should be connected to the center router on another LAN port. The AP is used to provide Internet connection to the wireless devices. If the number of IoT devices becomes large, the ports on the center router will be used out. To address this problem, a network switch or some modified routers can be used as LAN ports extension. We use off-the-shelf routers as LAN ports extension, by turning off their DHCP function and manually assigning an unused IP address within 192.168.0.0 - 192.168.255.255. The purchased IoT devices and emulated IoT devices can be connected to the testbed through the extended LAN ports and the wireless AP. The method of connecting commercial IoT devices to the test platform is the same as the method of using them in real life. We must register an account on the phone or web-page, activate the device, and then we can use its various cloud-based services and applications supported by the product manufacturer. In contrast, connecting the emulated IoT devices to the testbed is much simpler by treating them as Linux computers and connecting them to the same LAN.

Regarding the Mirai malware configuration, we already had a detailed discussion in Chapter 2. We establish the environment using the online Mirai code and following the online instruction. Then, we open a terminal to create bots, and use another terminal to log into the CNC server and send attack commands via telnet. When creating a bot or attacking a target, we constrain that only the local IP addresses between 192.168.0.1 to 192.168.0.255 are valid. In this way, all attack traffic is contained inside the local network while benign traffic for normal IoT operations (of the purchased IoT devices) can access the Internet. We will introduce more detailed configurations to ensure the safety of the testbed later in this chapter.

The advantage of the current testbed configuration is that the main router is a center point, and all network traffic would go through it. In this case, network traffic from single devices and the interactions among all the devices could be gathered by the main router, which sends the captured traffic to the monitoring PC. This rendezvous point greatly facilitates traffic collection and monitoring. In addition, this testbed enables malware experiments with guaranteed security, scalability, and

accessible data collection for network experiments. The following sections will explain how the above features are implemented in more detail.

4.2 Testbed Configuration

4.2.1 IoT Devices in the Testbed

Currently, there are 24 IoT devices attached in the testbed, as follows:

Table 4.1: List of IoT Devices.

Connection	*Category*	*Model, and service*	*LAN IP address*
Wired	Webcam	Dlink DCS 932L	192.168.0.12
		Dlink DCS 932L	192.168.0.134
	Network-attached storage	Synology NAS	192.168.0.163
	Router	GL-AR750S	192.168.0.1
		TPlink TLR480T	192.168.0.2
		Dlink Dl624	192.168.0.3
		Dlink Dl624	192.168.0.4
		Dlink Dl624	192.168.0.5
		Tp-link Archer C7 V5	192.168.0.6
		Tp-link Archer A6 AC1200	192.168.0.7
	Wireless access point	Unifi UAP AC pro	192.168.0.158
	Emulated IoT devices	Emulated webcam 1	192.168.0.127
		Emulated webcam 2	192.168.0.109
		Emulated temp sensor	192.168.0.137
		Emulated video stream	
		Emulated audio stream	192.168.0.139
		Emulated text stream	
Wireless	Door/window sensor	Braumm door/window sensor	192.168.0.119
		Globe door/window sensor	192.168.0.188
	Webcam	Dlink DCS 5020L	192.168.0.106
		Dlink DCS 5020L	192.168.0.134
		Littlelf LF-P1t	192.168.0.199
	Power plug strip	TPlink Kasa power strip	192.168.0.157
	LED light bulb	Smart LED Bulb	192.168.0.186
	Wireless Heater	Atomi smart Wifi heater	192.168.0.159
	Voice assistant	Amazon echo dot	192.168.0.177

The emulated service for video stream, audio stream, and text stream are using same hardware. All the above devices are typical IoT devices used in our daily life. In the testbed, there are seven wired IoT devices and seven wireless IoT devices. Besides devices' name and category, the LAN IP addresses, which can be found in the router's LAN DHCP list, are also important for identifying the devices. With different

connection approach, especially the wireless connection, we can attach more IoT devices in the testbed, which highlights the scalability. For example, we can purchase more IoT devices with wireless connection, or we can use the wireless connection feature from the Raspberry Pi board to emulated more IoT devices.

4.2.2 Traffic Mirroring

One of the features of this testbed is the easy collection of experimental data. By setting up the router and using Wireshark, every packet transmitted in the testbed can be captured in real-time. The data can be saved as pcap files. The real-time monitoring of the data streams and the saved data can be used to test the performance and accuracy of the network anomaly monitoring system and train algorithms and machine learning models.

In a regular network environment, a computer normally does not have access to all the packets being transmitted in a network for technical reasons and security/privacy considerations. Since this testbed needs to monitor and collect all data within the platform, we need to carefully configure routers to achieve this goal. Nowadays, some routers have the function of traffic mirroring/port mirroring. The mirroring function makes a copy of each passing packet and forwards the copy to a specified IP address or Ethernet port on the router.

There are two approaches to achieve traffic mirroring. The first approach is using a router with OpenWrt, an embedded Linux-based operating system for routers. Once the router is set up, we can use SSH to log into the router as the *root* user, as shown in Figure 4.2, where the model of the router is GL-AR750S-EXT and the OpenWrt version is 19.07.7.

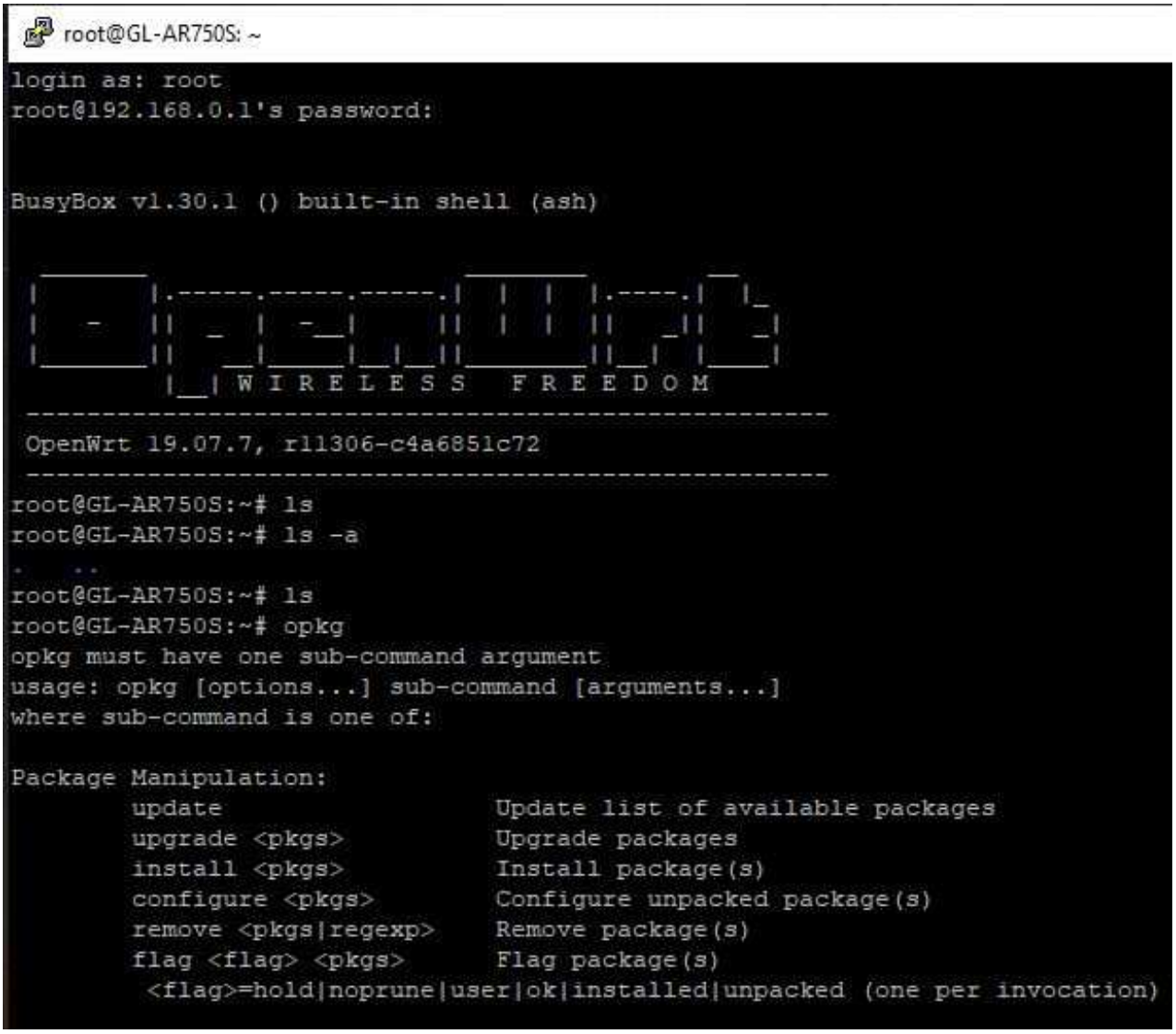


Figure 4.2: OpenWrt interface of router GL-AR750S-EXT.

Having the *root* access of the OpenWrt system, we can run two commands to set the *iptables* rules, as shown in Figure 4.3. The function of these two commands are copying and forwarding network packets from a source IP and a destination IP to a specified IP. The example in Figure 4.3 shows that all the network packets send from or send to IP address 192.168.0.241 will be mirrored to IP address 192.168.0.159.

```
root@GL-AR750S:~# iptables -A PREROUTING -t mangle -i br-lan ! -d 192.168.0.241
-j TEE --gateway 192.168.0.159
root@GL-AR750S:~# iptables -A POSTROUTING -t mangle -o br-lan ! -s 192.168.0.241
 -j TEE --gateway 192.168.0.159
```

Figure 4.3: Commands to set *iptables* rules.

After completing the setup of the router with the above method, we try to mirror the network packets of the device with IP address 192.168.0.241 to the computer with IP address 192.168.0.159. As shown in Figure 4.4, the packets that the device of IP address 192.168.0.241 receive from other IP addresses or the packets that the device sends to other addresses were successfully mirrored to the our monitoring computer (IP address 192.168.0.159). This validates the effectiveness of the first approach.

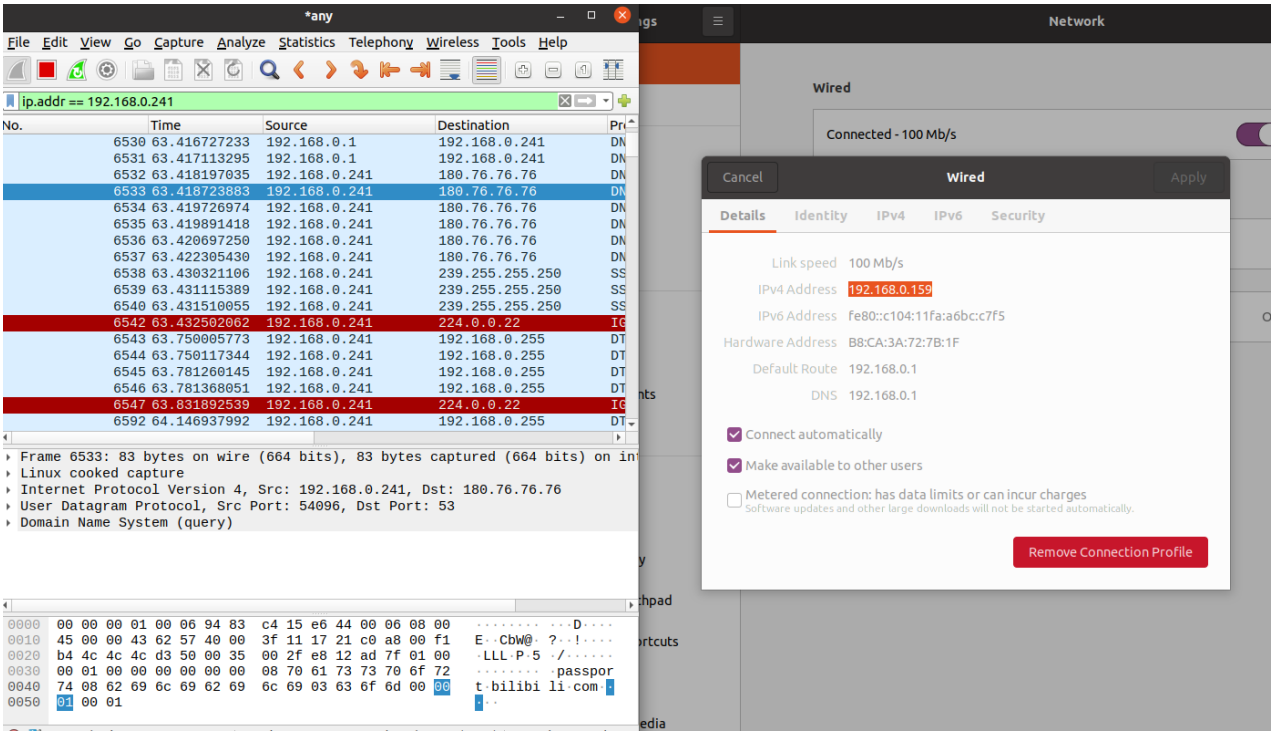


Figure 4.4: Sample pcap file showing the success of traffic mirroring.

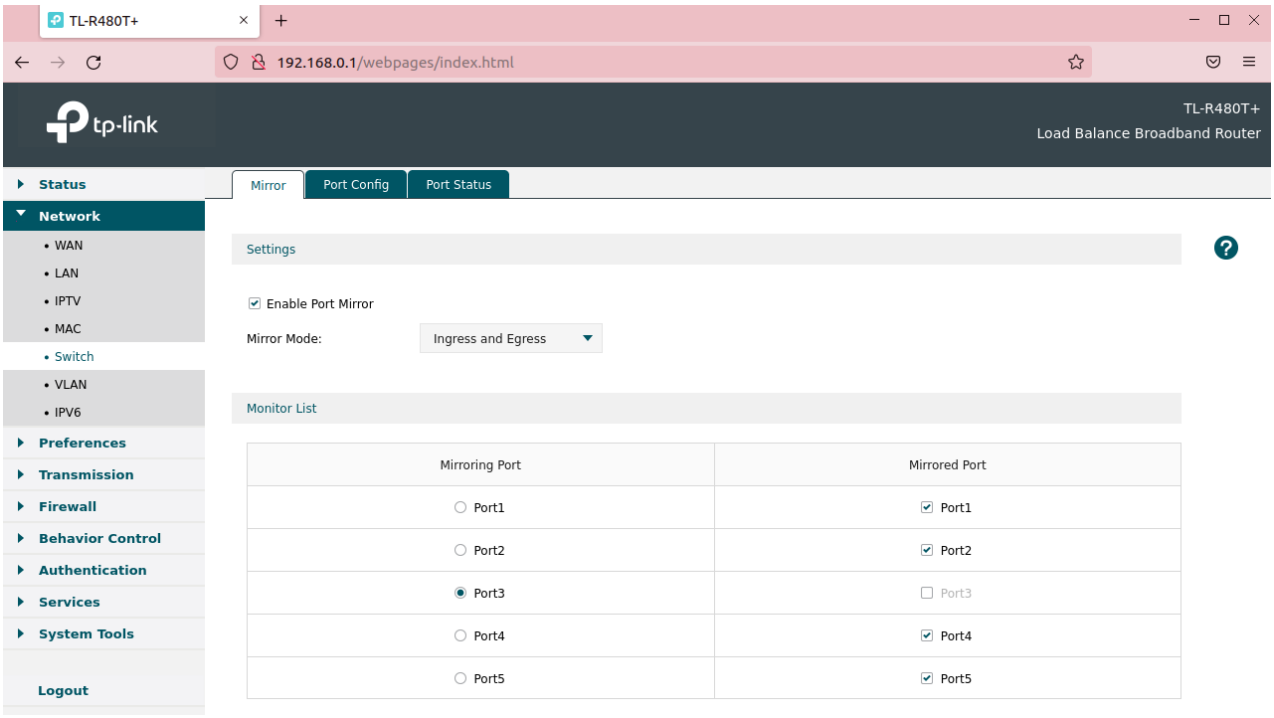


Figure 4.5: Interface of router's GUI system for port mirroring.

The second approach is to configure the router using the GUI provided by the device manufacturer, as shown in Figure 4.5 where the model of the router is tp-link R480T. We can access the GUI system in the browser and enable the port mirror function. In this case, all the data traffic through Ethernet port 1, port 2, port 4 and port 5 will be mirrored to the port 3. Therefore, after finishing the configuration, we just need to attach the monitoring computer to Ethernet port 3 on the router, then we can monitor and collect all the other Ethernet ports' network traffic. It is worth noting that not all routers provide this GUI support.

In practice, the router with mirroring function will be placed in the center of the network, and all the devices will attach on it. The attachment method could be direct attach to the Ethernet port, or attach through extended Ethernet ports or wireless access point. With the network traffic mirroring function, we can achieve real-time traffic monitoring over the monitoring computer. It is convenient and useful for real-time network anomaly detection system, because we can collect and observe all data packets in the testbed over one machine. The large amount of collected data can be used for training and evaluating anomaly detection models.

4.2.3 Security and Safety Measures

Since this testbed will be used to test and reproduce malicious code attacks, the security and safety of the platform becomes a serious concern. Here security means that the testbed is protected from outside attacks, and safety means that the testbed will not attack any machine outside of the testbed. Since the testbed is behind the two-level protection of the firewalls and intrusion detection systems (IDS) of the university and the Faculty of Engineering, the security of the testbed is automatically ensured by the above protection. Hence, our main focus is to guarantee the safety of the testbed.

The malicious code currently used for testing comes from open-source GitHub, and is open source. This open-source, research-purpose malicious code, however, is extremely complex and retains its original danger of attacking the Internet. To ensure the safety, we eliminate the threat of malicious code to environments outside the testbed with three approaches: (1) isolating the malware in virtual machine, (2) modifying malware by removing the automated part of the malicious code, and (3) building firewall between the testbed and the external network.

Isolating Malware in Virtual Machine

A virtual machine (VM) can provide an isolated environment for the malicious codes and the control center. In this thesis, we use a PC as the host machine and use VMware virtual machine to load the Mirai code and CNC. The configuration is listed in Tables 4.2 and 4.3. After completing the VM's setup and loading Mirai's code and CNC on the VM, we can modify and control Mirai in this environment via the interface shown in Figure 4.6.

The advantage of using a virtual machine is to prevent malware from attacking and infecting other parts of the computer, because we cannot be sure what will happen when malicious code is downloaded, unpacked, configured, and used with its control center. Also, virtual machines are subject to strict limitations on the scheduling of resources, drivers, or other software on the host machine. Once the malware becomes uncontrollable, or the system crashes, we can simply reset the virtual machine as easily as using a normal application.

Table 4.2: Information of the host machine.

CPU	Intel Core i7-2600 @ 3.4GHz
RAM	4 GB
Operating system	Windows 10 Education 21H2 19044.1706

Table 4.3: Information of the virtual machine.

Version	VMware Workstation 16.x Pro
Loaded OS	Ubuntu 18.04.3
Assigned RAM	2 GB
Assigned Storage	100 GB
Network	Bridged from host machine

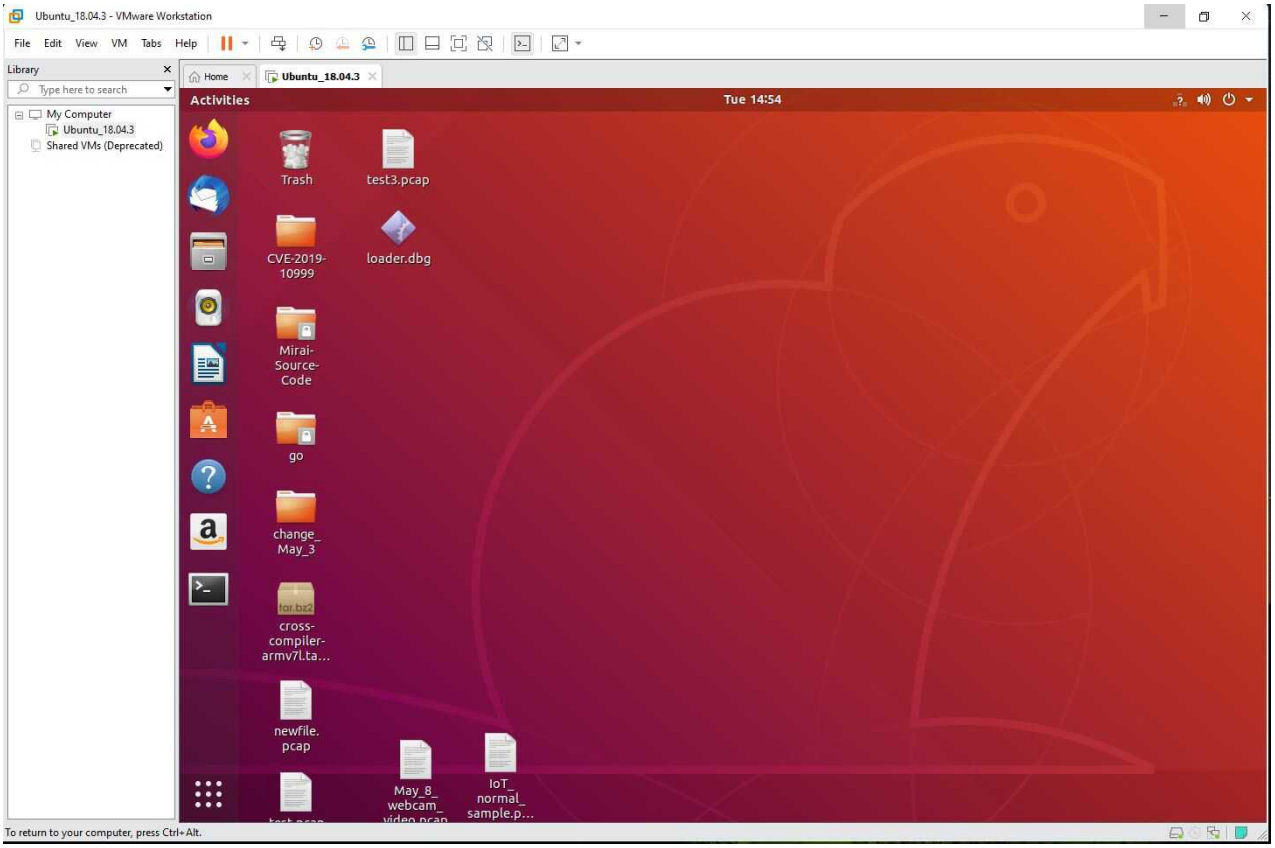
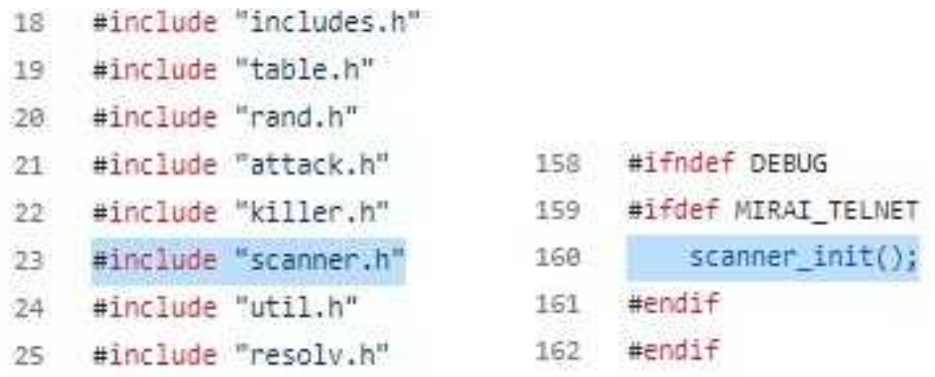


Figure 4.6: Interface of the virtual machine.

Modifying Malware

The danger of malicious code lies in its potential to sweep the surrounding environment automatically. Once an attack opportunity is discovered (e.g., an open port on a device), the malicious code will automatically load. This automatic feature makes it hard to control the malicious behaviour of Mirai. To avoid the problem, we study the open source code of Mirai and identify its automatic sweeping and loading functions, which are coded in its source files *scanner.c* and *scammer.h*. These two files implement the function of sweeping devices in a network and finding their vulnerabilities. To make the testbed safe, we can disable these two files from the *make* file. Another approach is to comment out the scanner functions in *Mirai-Source-Code/mirai/bot/main.c*, which include two parts as shown in Figure 4.7. After commenting out the scanner functions, compiling the Mirai code, and creating bots, the bots will not automatically scan the network.



(a) Scanner header file. (b) Initialization of the scanner.

Figure 4.7: Modified part in malicious code.

Building Firewall

Another safety measure is to create a firewall between the testbed and the external network. No matter what happens inside the testbed, the firewall will ensure that the attack packets will not leak to the external network. We use a router loaded with OpenWrt to configure the firewall. The WAN port on this router connects to the Internet. This router shares the Internet with the central router. In the OpenWrt system, we can set up the firewall rules by modifying the firewall file with the command shown in Figure 4.8. We then add new rules to the firewall file. For example, we can add the following rule shown in Figure 4.9, which means we block

all the outgoing data from the IP address 192.168.8.172 to the Internet, where this IP address refers to the Mirai compromised device. After the new rules are added, the firewall needs to be restarted Figure 4.10 to activate the new rules.

```
root@GL-AR750S:~# vi /etc/config/firewall
```

Figure 4.8: Command to open firewall file.

```
config rule
        option src 'lan'
        option name 'blockoutgoin'
        list src_ip '192.168.8.172'
        option dest 'wan'
        option proto '*'
        option target 'REJECT'
```

Figure 4.9: An example of new firewall rule.

```
root@GL-AR750S:~# /etc/init.d/firewall restart
```

Figure 4.10: Restart the firewall.

To verify that this method works, we try to link the external network with the restricted device. From Figure 4.11, we can see this device cannot access the external network. However, this devices can communicate with other devices inside the testbed without any problem, as shown in Figure 4.12.

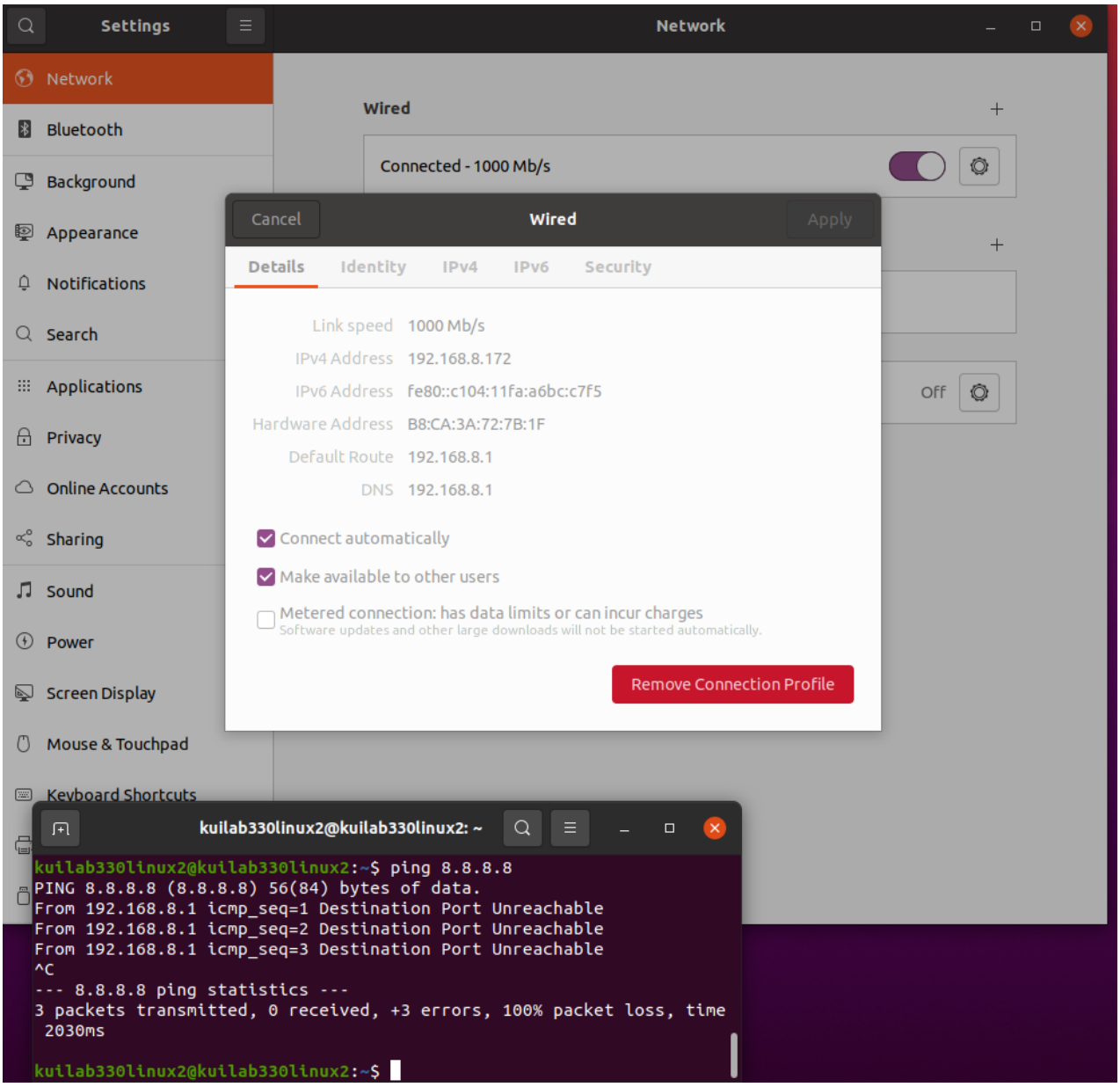


Figure 4.11: Ping external network.

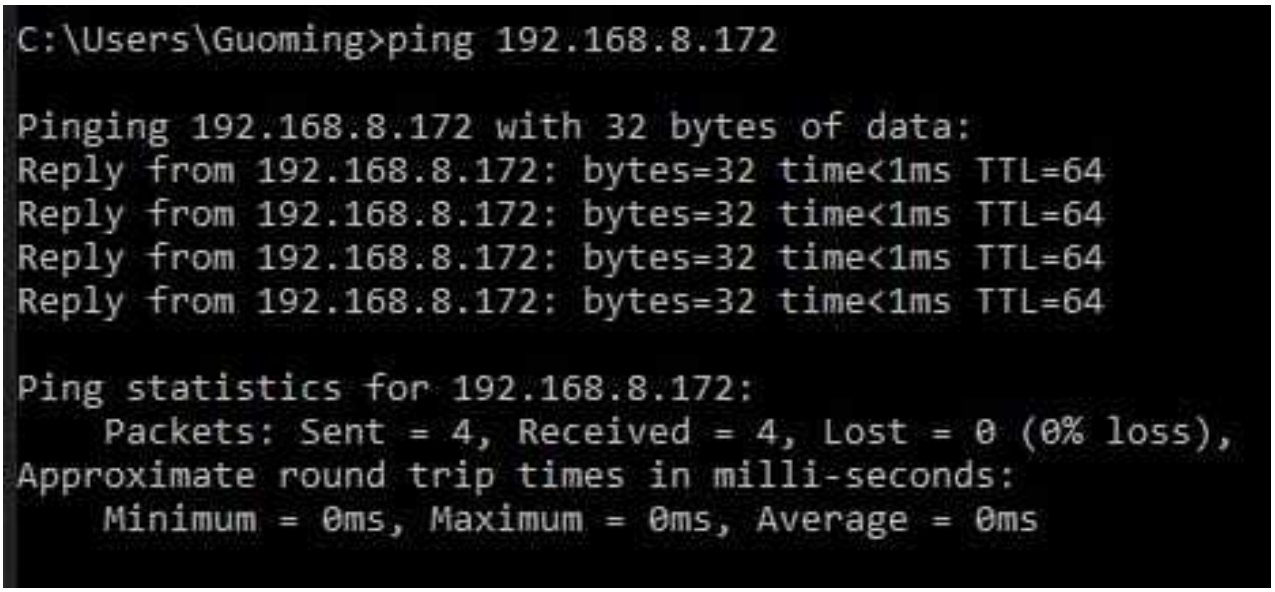


Figure 4.12: Ping within testbed.

It is worth noting that building firewall alone cannot ensure safety of the testbed, because the firewall does not enough intelligence to filter out only malicious traffic when a device must connect to the Internet to be functional. For instance, when we load Mirai malware to a purchased camera, this camera still needs to communicate to the Internet for its normal operation in the cloud (e.g., web-based video monitoring). In this case, we cannot block all outgoing packets from this device using the firewall, and we must depends on the other two measures to ensure the safety of the testbed.

As the last remark, we have run the testbed for nearly one year and never received any complaints/warnings from UVic IT support teams (department, faculty, and university levels) or any third-part entity. This clearly indicates that our testbed is safe.

4.2.4 Diverse Communication

The testbed provides different communication approaches, such as Ethernet and WiFi, to connect devices flexibly. The Ethernet connection provides wired links for more reliable communications. Since the number of Ethernet ports is limited on a router, we can use a network switch to extend the ports. To save cost, we in this thesis use some cheap off-the-shelf routers to extend ports. WiFi supports wireless communication, with which we can avoid the problem of the limited port number in a router. Nevertheless, WiFi is not as reliable as Ethernet cable.

Wi-Fi Protected Setup (WPS) is a network security standard used by many IoT devices. Since some IoT devices only support WPS communication, we set a wireless router with WPS to connect these IoT devices, and this wireless router is attached to the central router. In this thesis, we use the D-link DCS-5020L webcam, which supports Ethernet and WPS connection, as shown in Figure 4.13. To connect this IoT device, we use a router, tp-link Archer C7 V5, as the WPS wireless access point, as shown in Figure 4.14.

Figure 4.13: WPS switch on webcam.

Figure 4.14: WPS switch on router.

Chapter 5

Data Collection and Analysis

5.1 Motivation for Collecting IoT Network Traffic Data

After successfully building the testbed, we can use it for our security-related research: developing/evaluating IoT anomaly detection systems. We need to generate IoT traffic data in different scenarios, including both benign and malicious cases, and use the labelled data to train a machine learning model for anomaly detection. For this, data is the key.

Currently, there are several IoT network datasets, such as IoT23 [49], Kddcup99 [57], and NSL-KDD [43]datasets, available to the public. These datasets contain a diverse set of IoT network data and benefit security research by alleviating researchers' enormous burden of data collection. Nevertheless, we have found that many IoT scenarios have no labelled data in those datasets. For instance, when we plan to build context-aware anomaly detection, we need to distinguish between traffic pattern changes triggered by unreliable wireless links or by malware attacks; when we want to build a content-aided anomaly detection system, we need to label not only the packet header but also the content in the payload. In these cases, no existing dataset can meet our special research needs.

During the process of reproducing various scenarios, we can learn the causes of anomalies and the methods of attacking malicious codes. Moreover, having first-hand experience with the attack process can help us discover the vulnerabilities of IoT devices and develop targeted approaches to improve defences.

Due to the above reasons, data generation and collection over the testbed under

our full control are necessary for our ongoing and future studies.

5.2 Overview of Collected Data

5.2.1 Status

Until June 20 2022, there have been 24 IoT devices and 3 desktop machines in the testbed. The summary of the captured data is in Table 5.1:

Table 5.1: Captured data summary

Main category for captured Data	*Quantity*
Total number of pcap/pcapng files	112
Total file size	19.8 GB
Total duration time	$>$ 67 Hours
Total number of local devices involved	26
Total number of locations	2 (office and home)

After each IoT device was set up and started to generate traffic, we used Wireshark to collect the data flow copied by the port mirror function from the central router (Chapter 4). We need to use IoT devices to emulate different scenarios. For example, when a webcam works, most of time it just sends a fixed and small amount of packets periodically to indicate its connection state in the network. To make it generate a large amount of data traffic, we need to log in to the device's client web page/application to start live video mode.

Labelling data is an essential step for our research involving deep machine learning models. After capturing network data in Packet Capture (pcap)/PCAP Next Generation (pcapng) files, we record the following features for each captured file for easy file management:

- ***File name*** The name of the file.
- ***Start time*** The UTC clock time when capture started.
- ***Duration*** The duration between capture start and stop, in seconds.
- ***Number of packets*** The total number of packets in a file.

- ***File type*** The type of file, such as .pcap or .pcapng .
- ***File size*** The size of a file, in megabytes(MB).
- ***Date*** The date of file saved, in UTC date.
- ***Location*** The location of file captured.
- ***IoT devices involved*** List of devices attached in the testbed, when the data was capturing.
- ***Anomaly condition*** If contains malware, add the malware's name. If the problem is weak signal, add "Weak-signal". If normal, add "Normal".

5.2.2 Scenarios

For anomaly detection and classification, we classify IoT traffic according to different scenarios. So far, we have emulated and reproduced eight application scenarios listed in Table 5.2:

Table 5.2: Data collection by scenarios

Main category for data sample	*Detailed classification*
Normal working condition	1. Including few error packets
Abnormal w/o attacking	2. Network congestion
	3. Different locations
	4. Weak wireless signal
Abnormal under malware attack	5. Vulnerability exploitation
	6. Loading
	7. Communicating
	8. DDoS attacking

The normal scenario is created when the target IoT devices are working properly. There are very few error packets captured due to random changes in the network environment, e.g., a few duplicate packets due to re-transmission. The normal scenario would not trigger an evaluated anomaly detection model to report an alarm.

In the category of abnormal data without attacks, three scenarios may trigger abnormal events (i.e., the anomaly detection model under evaluation may report

anomaly): (1) network congestion, (2) weak wireless signal, and (3) model transfer (i.e., transfer of a detection model trained at one place to another place).

1. Network congestion: we let the webcam stream live video and other local machines transmit large files simultaneously. For example, we use one local PC to download multiple big files and open multiple browsers to play HD video.

2. Weak wireless signal: this scenario can be created by wireless IoT devices. When the wireless devices are far from the wireless access point or behind a metal-shielded place, the captured data may include a large amount of re-transmitted packets, which may trigger the anomaly detection model to report an anomaly.

3. Model transfer: This scenario is to test whether or not type-based detection model can be used robustly in different places. It has been assumed that same type of IoT device may generate similar traffic pattern, and as such, if we build an anomaly detection model for this IoT device type, the detection model can be used even if the IoT device works in different places [42]. To validate this assumption, we have collected data generated by the same type of webcam from two different locations: home and office.

There are four scenarios for attack traffic according to different stages of the malware attack: exploitation, loading, communicating, and attacking. The features of these four stages are different. In the exploitation stage, we use the open-sourced code to get root access on an IoT device, such as CVE-2019-10999. This stage has a short time duration and creates a small amount of packets. In the loading stage, the CNC server transmits the Mirai code to the target IoT device, and the dataflow is massive. In the communicating stage, the bot sends a small amount of data to declare its connection states and the CNC server sends the bot some commands. In the attack stage, the bot will generate massive and duplicate data to attack a target. We label data in different attack stages, which is useful for root cause analysis in anomaly detection research.

5.3 Statistical Analysis

This section includes the statistical analysis of collected data. Only the representative data and the data with significant features will be analyzed. The analysis is based on two dimensions: the number of network packets and data size (Bytes) transmitted in

a specific time interval. The time interval may vary in different scenarios based on the visualization of charts. Note that this statistical analysis is only meant to offer some preliminary results on the traffic patterns. How to utilize the analytical results to build or improve anomaly detection systems is beyond the scope of the thesis. In the following examples, each figure represents the data traffic from one device.

Benign Data

The benign scenarios' datasets for IoT devices have a clear characteristics. For instance, the traffic pattern from a webcam shows clear periodicity. Figure 5.1 and 5.2 show the collected data from a webcam with live video transmission on. Both the number of packets (Figure 5.1) and the size of data (5.2) confirm such periodicity. This phenomenon can be explained that IoT devices are special purpose devices. Unlike general purpose computers, IoT devices are for simple usage and thus generate traffic of salient features. People can use this feature to develop the anomaly detection system which can identity the IoT network traffic and separate the malicious data.

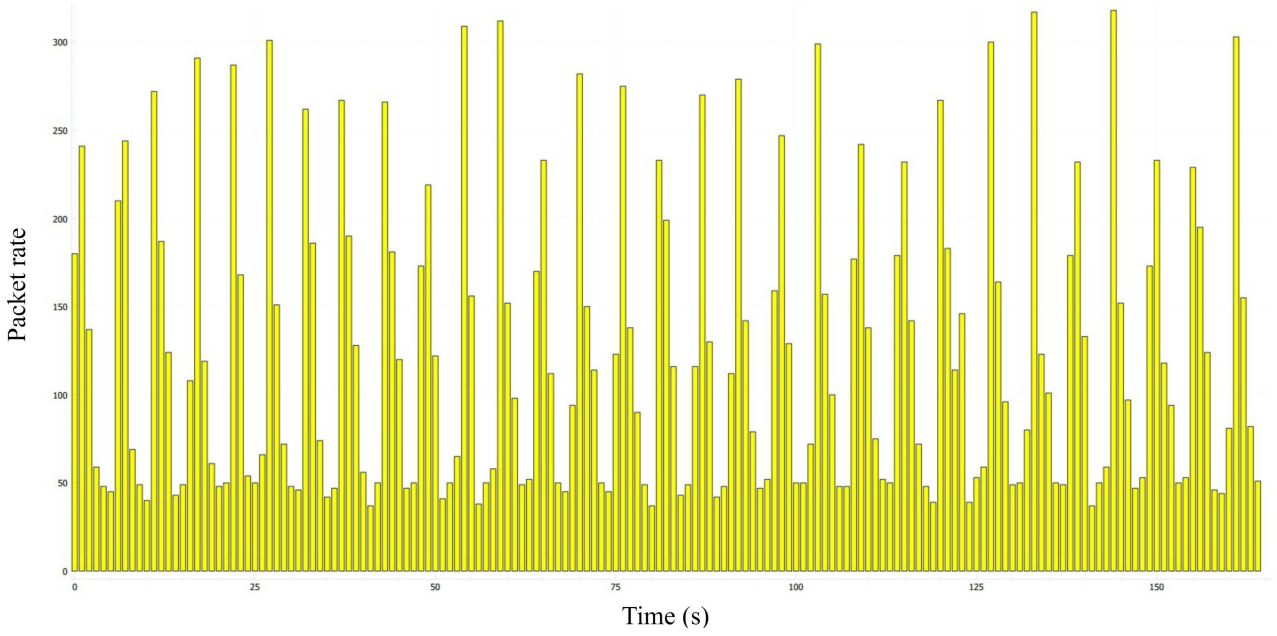


Figure 5.1: Webcam working with live video on, number of packets in time interval of 1 second.

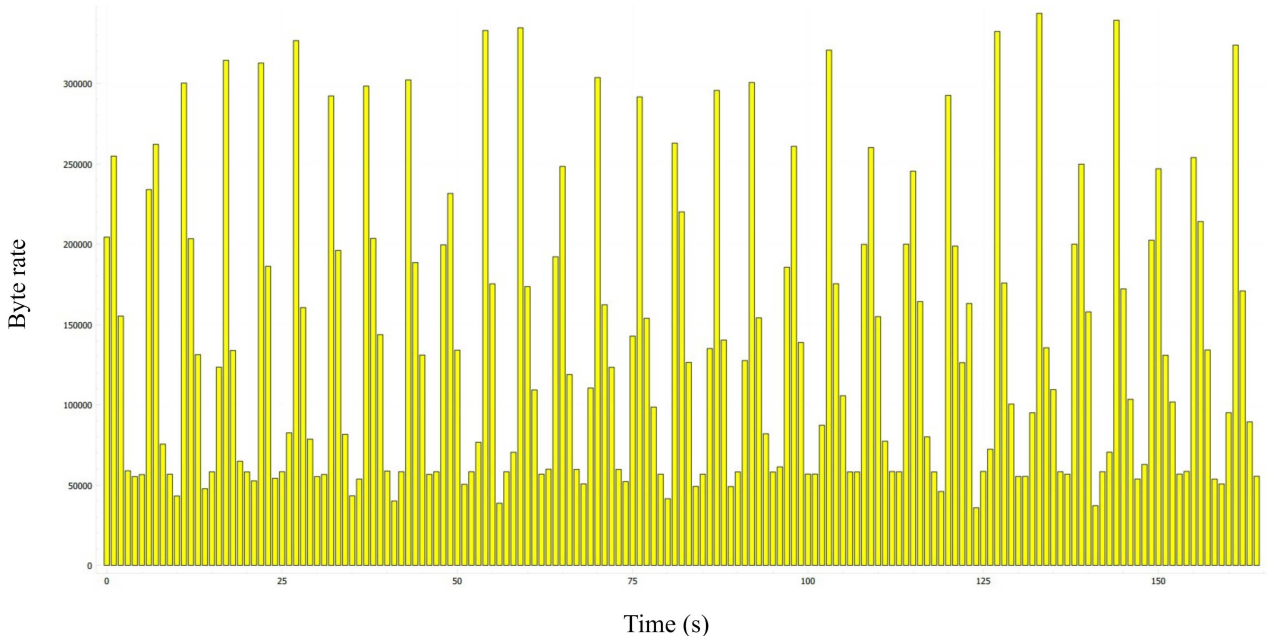


Figure 5.2: Webcam working with live video on, data size (Bytes) in time interval of 1 second.

Different Location/Network Environment

In practice, the same type of device is used in different places, and the parameters of the same network environment may change over time and with external influences. For example, a university lab and a residential home may have different network bandwidth and network delay. The number of devices that share the same network with the IoT devices can also significantly impact the traffic of IoT devices. The above considerations raise the need to validate the robustness of type-based IoT anomaly detection models [42].

Figure 5.3 and 5.4 show the data collected from our lab at UVic. Figure 5.5 and 5.6 show the data collected from a residential house. All the data were generated by the same webcam, i.e., same device with identical device configuration (except IP address). Clearly, the data from the lab has a larger size (in Bytes) transmitted than residential data. In addition, the lab data shows higher volatility than the residential data.

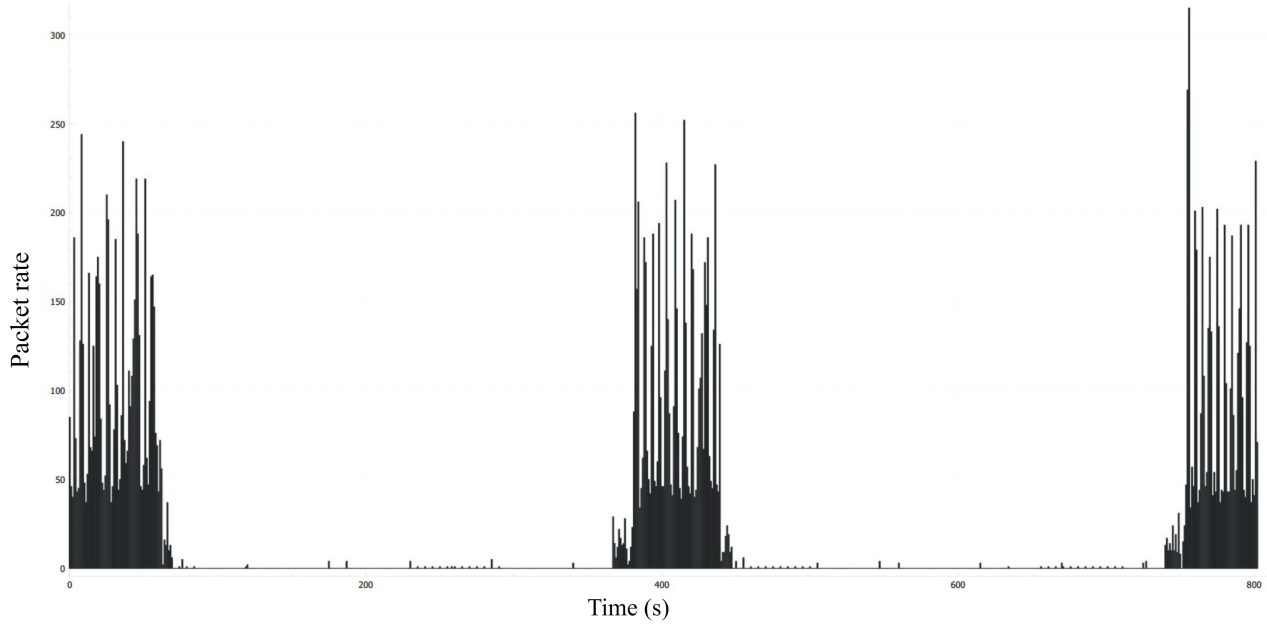


Figure 5.3: Webcam data collected in the lab, number of packets in time interval of 1 second.

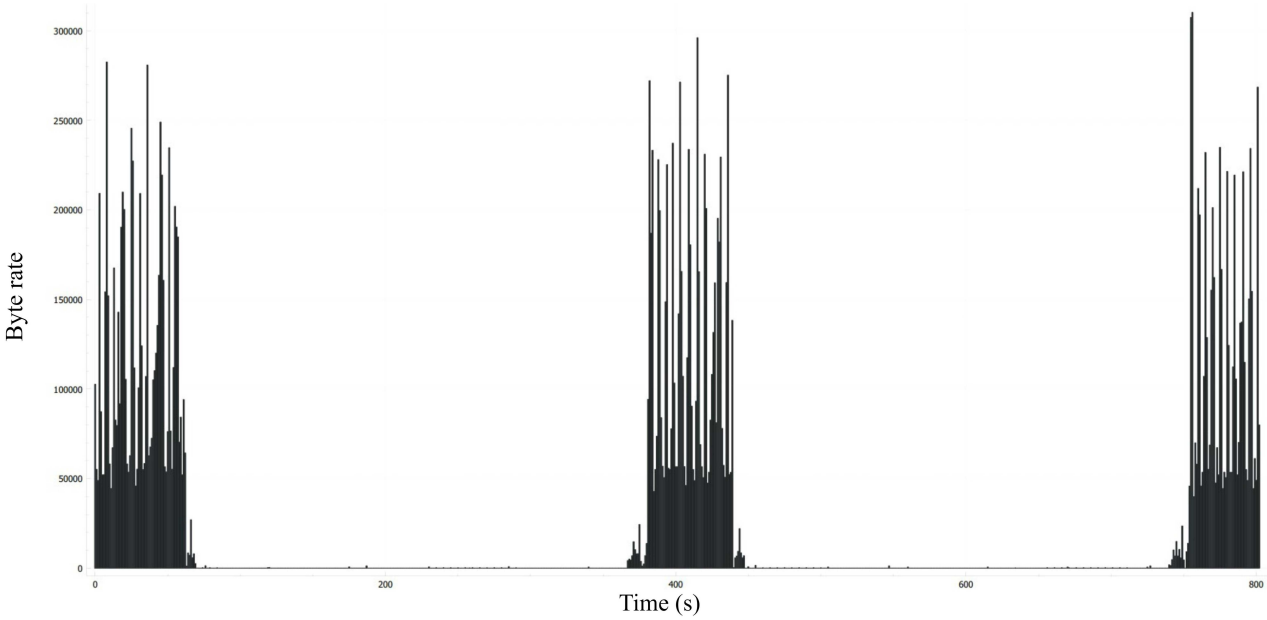


Figure 5.4: Webcam data collected in the lab, data size (Bytes) in time interval of 1 second.

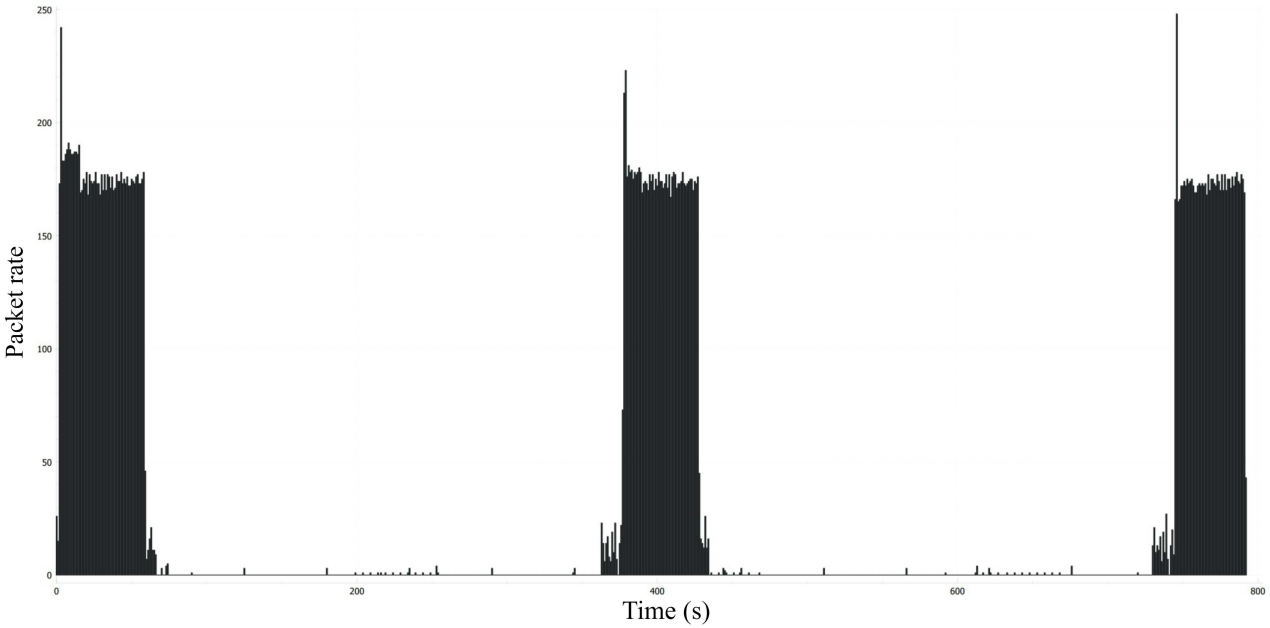


Figure 5.5: Webcam data collected in a residential location, number of packets in time interval 1 of second.

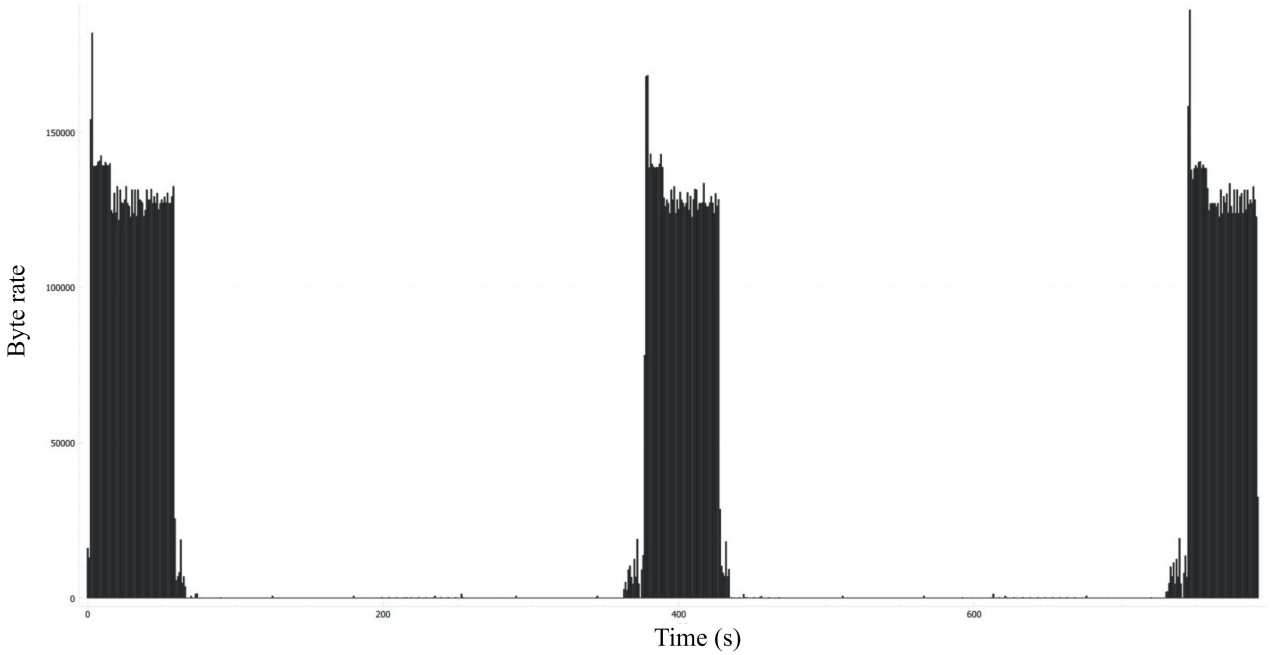


Figure 5.6: Webcam collected in a residential location, data size (Bytes) in time interval of 1 second.

Network Congestion

Network congestion may impact the performance of an anomaly detection model, e.g., the model may report false attacks. To test the robustness of a detection model in the presence of network congestion, we artificially generate different network conditions and collect the corresponding network traffic. To achieve the goal, we can use multiple machines to start multiple applications that consume high bandwidth (e.g., transfer large files or open multiple browsers to watch high resolution videos).

From Figure 5.7, we can see a remarkable contrast with the normal data shown in Figure 5.1. The impact of network congestion on the data flow of IoT devices is huge, and it makes the data flow of IoT devices lose its periodicity. When we watched the tested webcams, their videos were choppy or jerky.

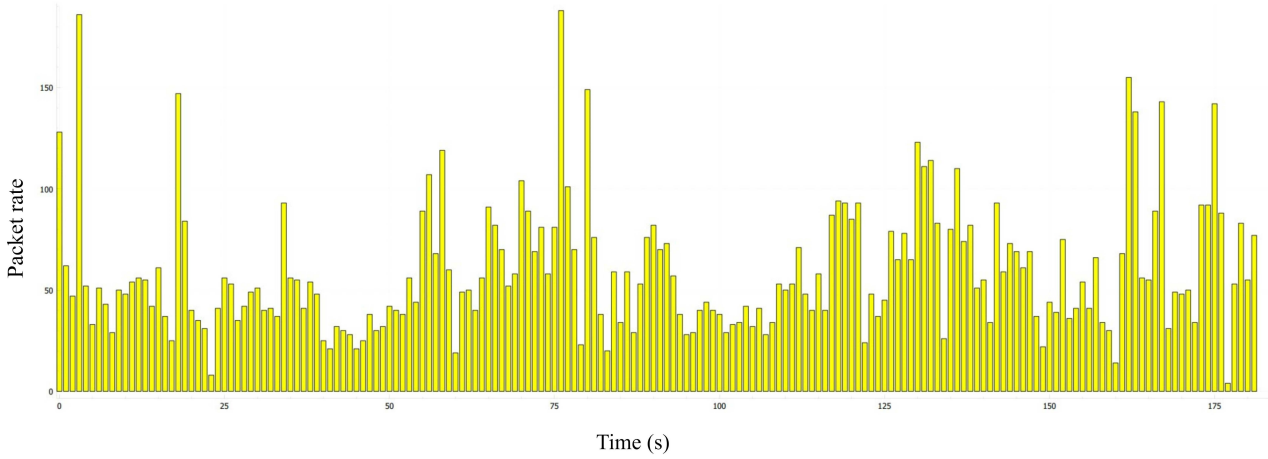


Figure 5.7: Webcam live video in the presence of network congestion.

Vulnerabilities Exploitation

In Figure 5.8, the red line represents attack traffic, and the blue line represents the benign traffic. The time interval is 0.01 seconds because this process is very short. Nevertheless, we can still find that the packets carried the attacking message even if the time duration is short. As shown in Figure 5.9, we can see the long value assigned to "WEPEncryption"", which leads to buffer overflow. Vulnerabilities exploitation is the first and an important step of attacking. Due to its short time duration, we may need to check the packet payload with deep packet inspection (DPI).

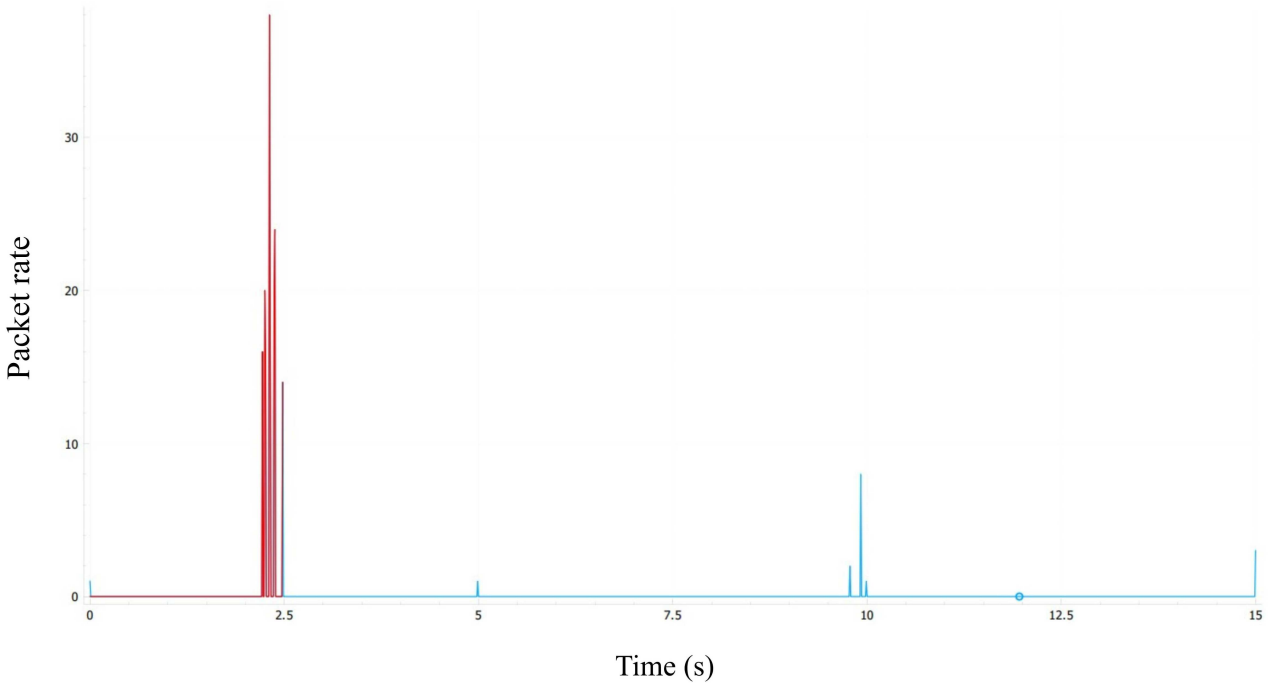


Figure 5.8: Traffic during vulnerability exploitation.

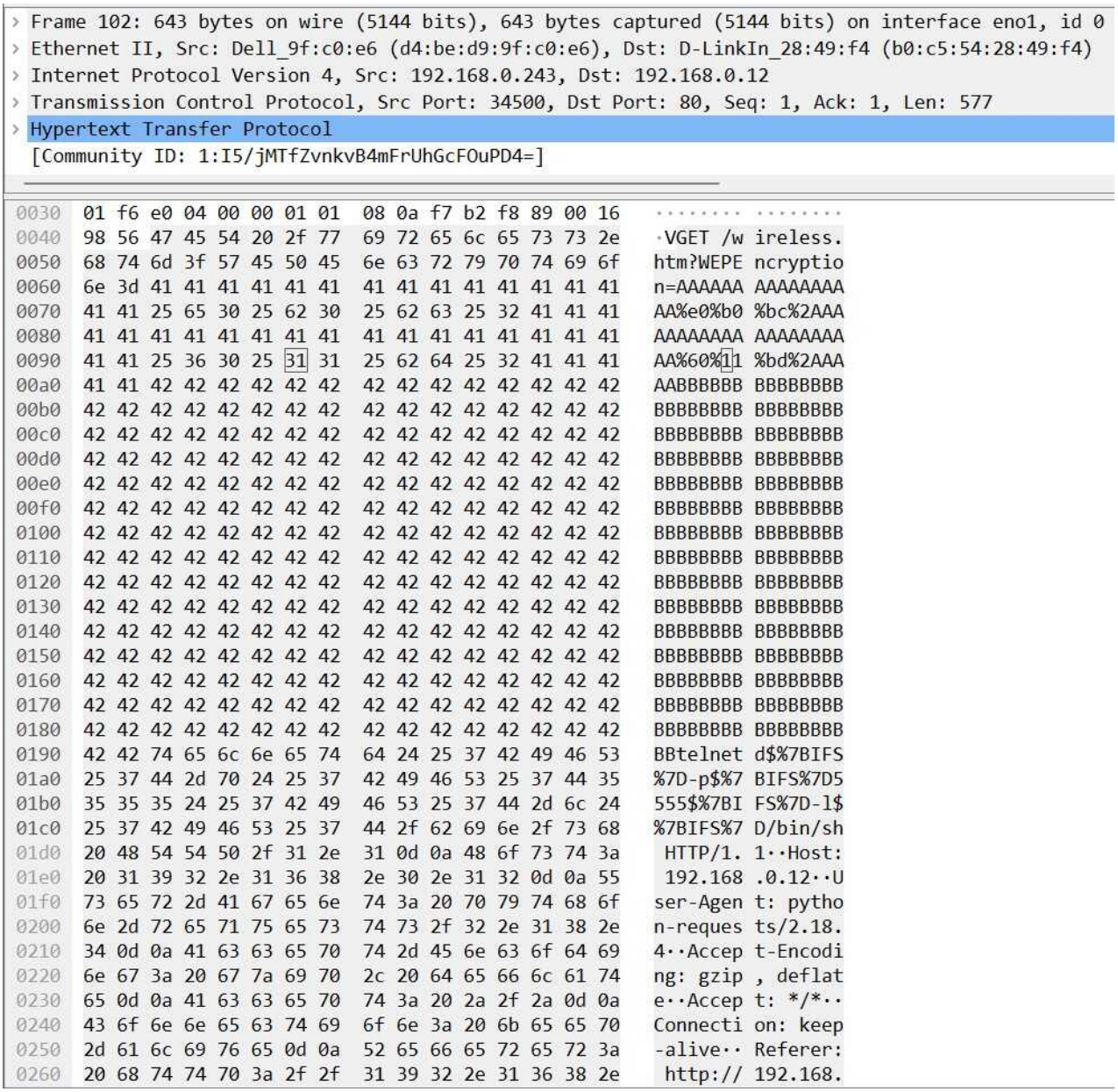


Figure 5.9: Payload of CVE-2019-10999.

Loading Malicious Code

After getting the root access to an IoT devices, we can create the bot by loading the compiled Mirai code onto the target device. The loading process will create data traffic with notable features. Figure 5.10 and 5.11 show the network traffic during the Mirai loading when webcam is in the live video mode. The red bar represents malicious traffic, and the yellow bar represents benign traffic. If the red and yellow

bar exist in same time interval, they do not overlap but add on each other, because the traffic is from the same device and each bar shows the total data traffic in a time interval. During Mirai loading, the number of malicious packets is much larger than the number of benign packets, and the process lasts for a while. After the loading, the data traffic went back to normal and periodic.

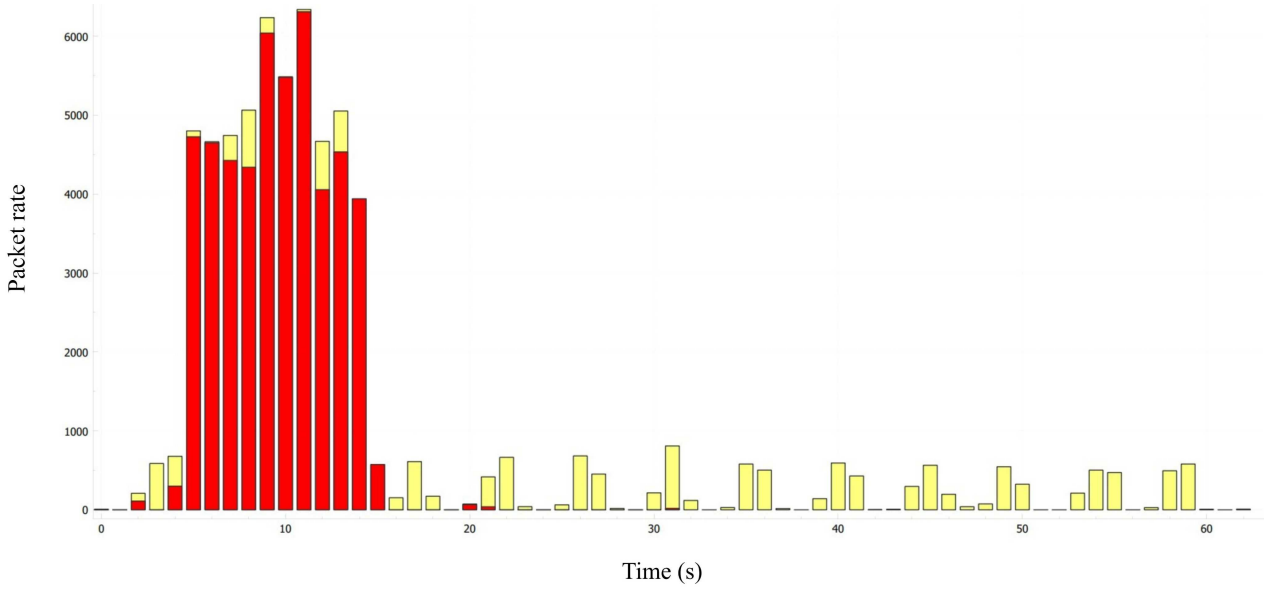


Figure 5.10: The Mirai loading process stands out w.r.t. the number of packets per second. The webcam is in the live video mode.

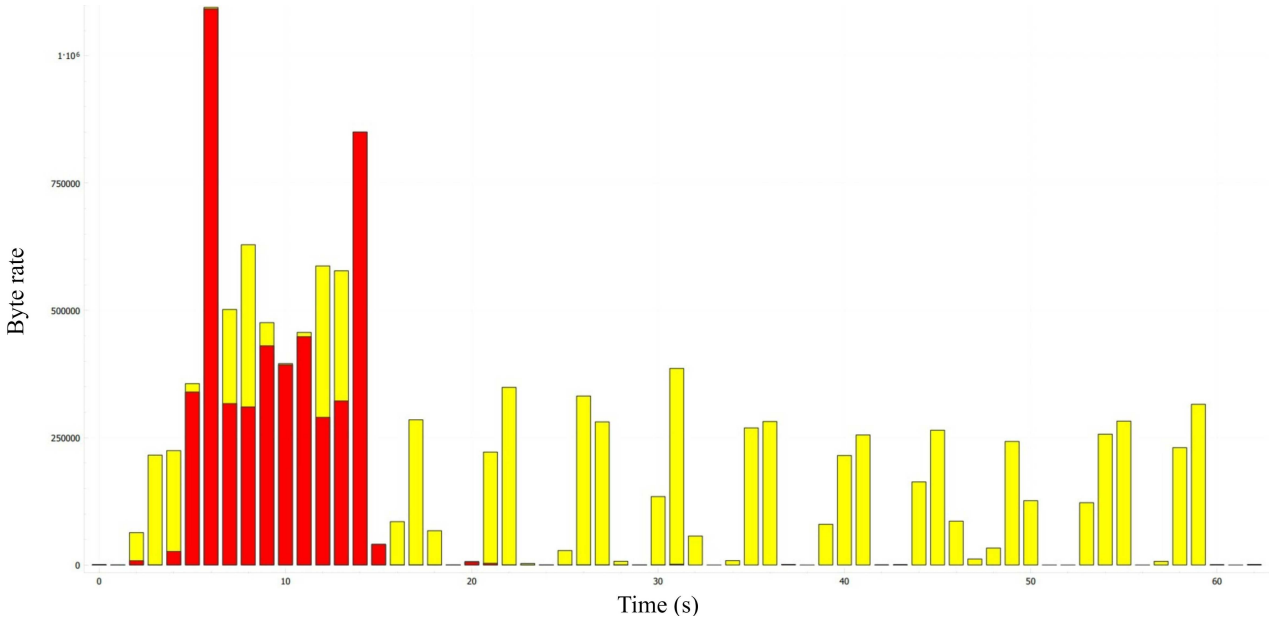


Figure 5.11: The Mirai loading process stands out w.r.t. with data size (Bytes) per second. The webcam is in the live video mode.

Bots Launch DoS Attacks

After creating the bots, the attacker can send commands to bots and control them for launching a DoS attack to other victims. A DoS attack generally sends a large number of network packets to a target, such as HTTP flood, UDP plain flood, TCP stomp flood, or DNS resolver flood, to drain the target's network resources and make their services unavailable. In Figure 5.12, a bot is sending UDP packets to a victim, which is another Linux machine in the testbed. The red bar represents malicious traffic, and the yellow bar denotes benign traffic. If the red and yellow bar exist in same time interval, they do not overlap but add on each other, because the traffic is from the same device and each bar shows the total data traffic in a time interval. The number of attack packets per second is small, but the traffic is consistent and periodic. Besides, the DoS traffic does not effect normal data significantly. It is worth noting that this phenomenon is because we only create one bot for the attack. In real-world DoS attacks, the number of bots could be thousands. From Figure 5.12, we can see that when the number of bots is small, the attack traffic may successfully hide in normal data, making the attack detection difficult.

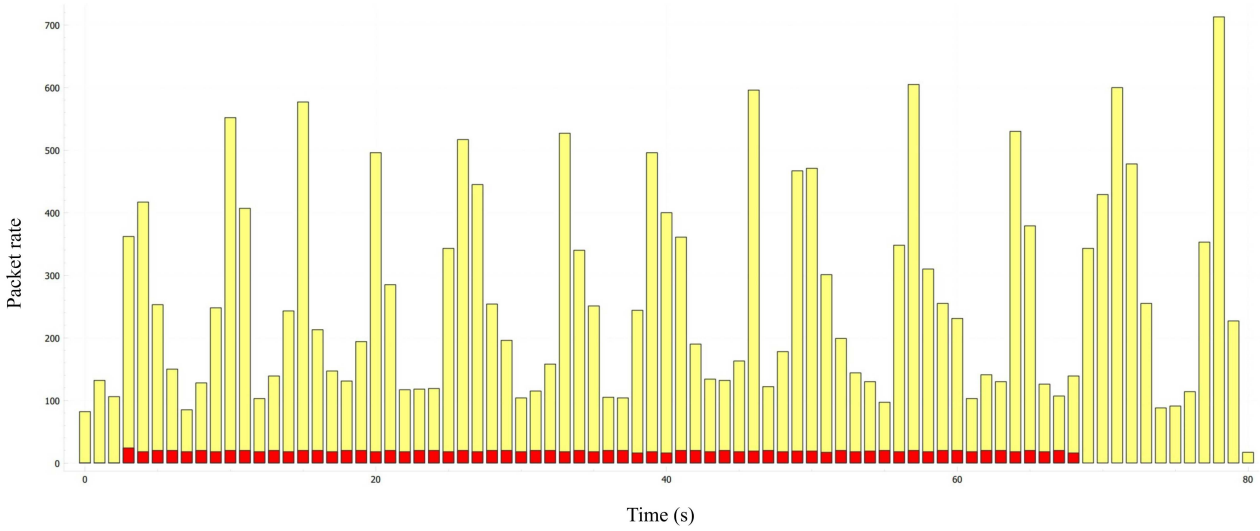


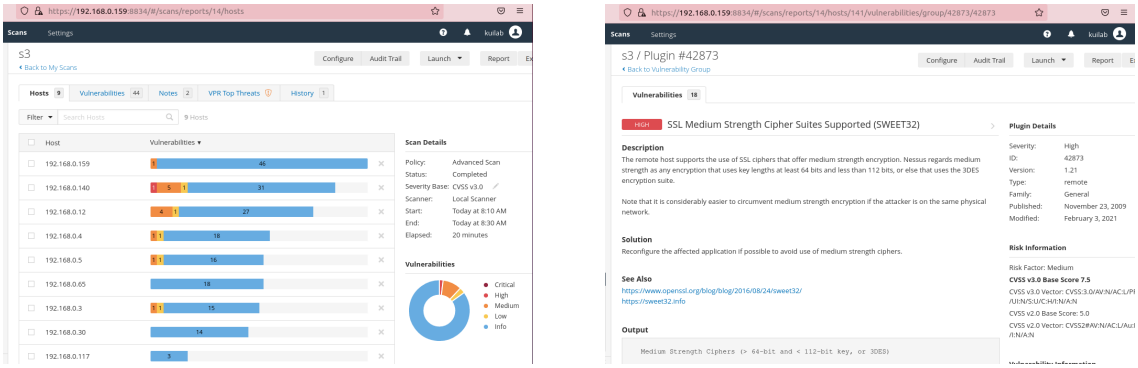
Figure 5.12: A Mirai-controlled webcam lunches DoS attack.

5.4 Device Security Analysis

As stated in Chapter 4, we mainly focus on the safety of the testbed, while the security of the testbed relies on the security protection of the university and the faculty levels. Nevertheless, we need to do our due diligence to ensure that the devices over our testbed have no exposed vulnerability to avoid outside attacks. For this purpose, we performed a scan and analysis of the security on all IoT devices within the testbed.

There are plenty of tools, such as Nessus or Nexpose, to achieve this work. We can also use a Kali Linux machine in the testbed. This OS comes with diverse scan and security test tools, such as Nmap, Nikto, or SQLMap. In this thesis, we use the Nessus to perform IoT security test.

Nessus can be installed on Linux and Windows machine, and it comes with GUI. Users can open the user interface in a browser as shown in Figure 5.13. Nessus sweeps the available ports and services on devices or web-servers. If there is any opportunity, it will lunch a simulated, non-harmful attack. All the sweepings and simulated attacks will be compared with the Common Vulnerability Scoring System (CVSS). Finally, it will generate a security report for the tested devices, which includes summary of each device's vulnerabilities, pie chart and bar chart for status, and detailed description of each vulnerability.



(a) Devices and vulnerability status.

(b) Single vulnerability description.

Figure 5.13: Sample Nessus scan GUI.

Nessus classified the risk into five levels: critical, high, medium, low, and minor. The minor risk is usually harmless. We can see that a large number of minor risks exist on the devices. The critical risk usually means a vulnerability that may cause a breach in confidentiality, integrity, and availability. We have not found any critical risk in our testbed. However, there are a certain number of high, medium, and low risks discovered in the purchased IoT devices. These risks might cause some degree of privacy breach or intrusion into the system. We list the found major vulnerabilities in Table 5.3.

Table 5.3: Major vulnerabilities on devices.

Main category	*Detailed classification*
High risk	SSL medium Strength Cipher Suites Supported (SWEET32)
	TCP/IP initial sequence number reuse
	SSL certificate signed using weak hashing
Medium risk	SSL self-signed certificate
	Certification expiry
	Certificate cannot be trusted
	IP forward enabled
	TLS 1.1 protocol deprecated
	TLS 1.0 protocol detection
Low risk	Short SSL RSA keys
	Web server uses basic Authentication w/o HTTPS
	Multiple Ethernet driver frame padding info disclosure
	DHCP server detection

Among the tested IoT devices, the new products of recent years and the products of large companies are very good in terms of security. They only have minor risks and a few low risks. The security problems are concentrated on older devices and some specific categories, such as webcams and routers listed in Table 5.4. This is because (1) older devices may lack security updates on their firmware, and (2) some specialized devices such as routers and webcams have a high chance of being directly exposed to the Internet. Overall, in the devices security test, we found that 81% the risks are minor, and 5% of them are low risk, 11% of them are medium risk, 3% of them are high risk, and none critical risks have been discovered yet, as shown in Figure 5.14.

Table 5.4: Devices with major vulnerabilities.

	High	Medium	Low
TL-R480T router	2	6	2
5020L webcam	1	8	1
932L webcam	1	4	1
Dl-624 router	0	1	1

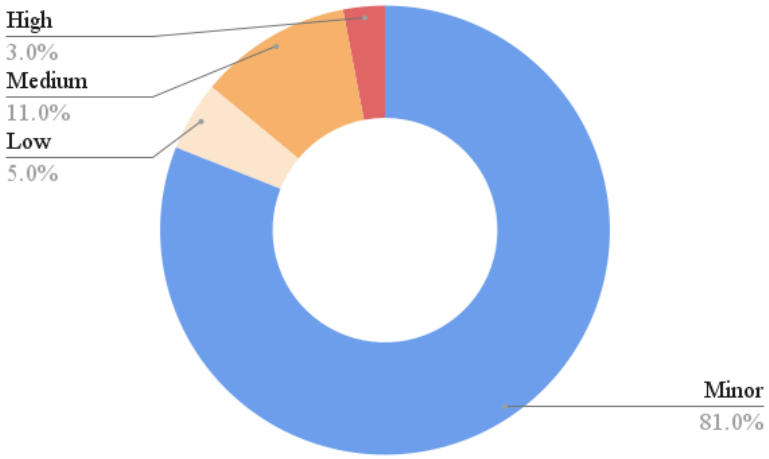


Figure 5.14: The weight of discovered vulnerabilities.

The above security analysis gives us enough confidence that our testbed is secure. It is worth noting that we cannot completely remove the discovered vulnerabilities because the device manufacturers do not provide any firmware update to fix the "problems". It is worth noting that the vulnerabilities, particularly the non-critical ones, do not necessarily mean an attacker can compromise the devices easily, considering that our testbed is also behind the two-level firewall protection of the university and the Faculty of Engineering.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

To collect customized data to meet our specific needs in studying IoT network anomalies and developing defence systems, we developed a safe and secure testbed that uses emulated IoT devices for cost-saving. In the meantime, building this testbed has helped us understand the working mechanisms of IoT devices, malware, and computer networks.

In this thesis, we first introduced the evolution of IoT and its importance to today's world. We also discussed the vulnerabilities of IoT devices and the security threats they face. Among the various issues, we took two common threats to IoT devices as examples, stack buffer overflow and Mirai botnet. By studying how they work in detail, we can reproduce the attacks, collect relevant data, and gain experience on how to defend against them.

In the process of developing the testbed, we took into account scalability, flexibility, safety, and security. To achieve these features, in addition to mirroring all network traffic to a central monitoring machine for easy data collection, we used a variety of special settings and multiple network connections to increase the diversity of devices.

We have identified two critical challenges in building such a testbed: (1) commercial IoT devices may be too costly if we want to expand the testbed, and (2) it is non-trivial to compromise a commercial IoT device and launch malware onto the device. To address the above difficulties, we built emulated devices with Raspberry Pi. By using Raspberry Pi, add-on devices, and different software, the emulated devices

are close to commercial IoT devices at the hardware, software, and behaviour levels. More critically, we have complete control over the emulation device, which allows us to load any code to the emulation devices and have the emulation devices deliver the content we want. Overall, emulating IoT devices not only grants us enough flexibility in controlling the devices' behaviour but also reduces the hardware cost significantly. It is much easier to scale up the testbed with emulated IoT devices.

For the safety of the testbed, we have set up a firewall at the gateway to the external network, changed the malware's source code, and isolated the malware in a virtual machine. The primary purpose of the above methods is to ensure that the malware does not create any adverse impact outside the test platform.

For the security of the testbed, we mainly rely on the two-level firewall protection at the university and the Faculty of Engineering levels. Nevertheless, we also performed a secure analysis on the devices used in the testbed to ensure no critical vulnerabilities exist in the testbed.

After building the testbed, we used it to reproduce a variety of application scenarios, including normal IoT operations, the abnormal state without malware attacks, and the abnormal state during attacks. We performed a preliminary statistical analysis of the collected network traffic. From the analysis, we can see that the packets transmitted by IoT devices usually exhibit periodicity regarding the number of packets and the data amount. Various conditions, some malicious and some not, can break this regular pattern. Accordingly, we have reproduced different scenarios that may trigger an anomaly detection system to (false) alarms. The labelled data from different scenarios will be beneficial for enhancing IoT anomaly detection in a targeted way in future work.

Finally, it is essential to emphasize that our testbed is used only for academic research and security systems development. Although open-source malicious code has been used in the testbed, we have taken every security measure imaginable to ensure that it does not cause any damage outside the testbed.

6.2 Future Work

In future work, this testbed's potential applications and extensions are diverse. Overall, there are two main directions. In the first direction, we will add more devices to the testbed, emulate more application scenarios, and collect more data. To scale up the testbed economically, we will use Raspberry Pi to emulate more IoT devices. This

approach will improve efficiency and save the cost of expanding the testbed. In addition, we plan to develop different generative models to generate and replay synthetic data over emulated devices. Based on the historical video data from a commercial webcam, it is possible to build a data generation model that creates synthetic data of similar statistical patterns.

The other direction is developing, deploying, and testing new anomaly detection systems on the testbed. Along this line, we plan to develop content-aided IoT anomaly detection system and context-aware IoT anomaly detection system. Since most IoT traffic is encrypted, we also plan to investigate the problem of identifying device types with encrypted network traffic.

Bibliography

[1] Erry Yulian Triblas Adesta, Delvis Agusman, and Avicenna Avicenna. Internet of things (iot) in agriculture industries. *Indonesian Journal of Electrical Engineering and Informatics (IJEEI)*, 5(4):376–382, 2017.

[2] Sarah A Al-Qaseemi, Hajer A Almulhim, Maria F Almulhim, and Saqib Rasool Chaudhry. Iot architecture challenges and issues: Lack of standardization. In *2016 Future technologies conference (FTC)*, pages 731–738. IEEE, 2016.

[3] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. Sok: Security evaluation of home-based iot deployments. In *2019 IEEE symposium on security and privacy (sp)*, pages 1362–1380. IEEE, 2019.

[4] KEVIN ASHTON. "that 'internet of things' thing".

[5] Javier Augusto-Gonzalez, Anastasija Collen, S Evangelatos, Marios Anagnostopoulos, Georgios Spathoulas, Konstantinos M Giannoutakis, Konstantinos Votis, Dimitrios Tzovaras, B Genge, Erol Gelenbe, et al. From internet of threats to internet of things: A cyber security architecture for smart homes. In *2019 IEEE 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 1–6. IEEE, 2019.

[6] Andreas Becker and Ing Christof Paar. Bluetooth security & hacks. *Ruhr-Universität Bochum*, 2007.

[7] Andrea Castillo O'Sullivan and Adam D Thierer. Projecting the growth and economic impact of the internet of things. *Available at SSRN 2618794*, 2015.

[8] Shaibal Chakrabarty and Daniel W. Engels. A secure iot architecture for smart cities. In *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 812–813, 2016.

[9] Wang Chengjun. The analyses of operating system structure. In *2009 Second International Symposium on Knowledge Acquisition and Modeling*, volume 2, pages 354–357. IEEE, 2009.

[10] CVE. Cve-2019-10999. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-10999#:~:text=The%20D%2DLink%20DCS%20series,htm.

[11] The New Jersey Cybersecurity and Communications Integration Cell (NJCCIC). Mirai botnet. https://web.archive.org/web/20161212084605/https://www.cyber.nj.gov/threat-profiles/botnet-variants/mirai-botnet.

[12] Cristiano André Da Costa, Cristian F Pasluosta, Björn Eskofier, Denise Bandeira Da Silva, and Rodrigo da Rosa Righi. Internet of health things: Toward intelligent vital signs monitoring in hospital wards. *Artificial intelligence in medicine*, 89:61–69, 2018.

[13] Danco Davcev, Kosta Mitreski, Stefan Trajkovic, Viktor Nikolovski, and Nikola Koteli. Iot agriculture system based on lorawan. In *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 1–4. IEEE, 2018.

[14] Larry Dignan. Internet of things big security worry, says hp, 2014.

[15] dilitrust. "cyber attacks on smart cities: Why we need to be prepared".

[16] Altaf Engineer, Esther M Sternberg, and Bijan Najafi. Designing interiors to mitigate physical and cognitive deficits related to aging and to promote longevity in older adults: a review. *Gerontology*, 64(6):612–622, 2018.

[17] Shuangquan Fu and Pritesh Chandrashekhar Bhavsar. Robotic arm control based on internet of things. In *2019 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pages 1–6. IEEE, 2019.

[18] fuzzywalls and secure 77. Cve-2019-10999. https://github.com/tacnetsol/CVE-2019-10999.

[19] Nick G. "how many iot devices are there in 2022? [all you need to know]".

[20] Arthur Gatouillat, Youakim Badr, Bertrand Massot, and Ervin Sejdić. Internet of medical things: A review of recent contributions dealing with cyber-physical systems in medicine. *IEEE internet of things journal*, 5(5):3810–3822, 2018.

[21] Kaustubh Gawli, Parinay Karande, Pravin Belose, Tushar Bhadirke, and Akansha Bhargava. Internet of things (iot) based robotic arm. *Int. Res. J. Eng. Technol*, 4(03), 2017.

[22] Dimitris Geneiatakis, Ioannis Kounelis, Ricardo Neisse, Igor Nai-Fovino, Gary Steri, and Gianmarco Baldini. Security and privacy issues for an iot based smart home. In *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1292–1297. IEEE, 2017.

[23] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[24] Sumeet Gupta, Christophe Meunier, and Jiva J. Jagtap. Telecom transformation agenda for the 2020s. `https://www.fticonsulting.com/emea/insights/articles/telecom-transformation-agenda-2020s`.

[25] Jasmin Guth, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Lukas Reinfurt. Comparison of iot platform architectures: A field study based on a reference architecture. In *2016 Cloudification of the Internet of Things (CIoT)*, pages 1–6, 2016.

[26] Siham Al Hinai and Ajay Vikram Singh. Internet of things: Architecture, security challenges and solutions. In *2017 International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions) (ICTUS)*, pages 1–4, 2017.

[27] Foundeo Inc. 2021 security vulnerability report. `https://stack.watch/stats/2021/`.

[28] S Jaiganesh, K Gunaseelan, and V Ellappan. Iot agriculture to improve food and farming technology. In *2017 Conference on Emerging Devices and Smart Systems (ICEDSS)*, pages 260–266. IEEE, 2017.

[29] Yin Jie, Ji Yong Pei, Li Jun, Guo Yun, and Xu Wei. Smart home system based on iot technologies. In *2013 International conference on computational and information sciences*, pages 1789–1791. IEEE, 2013.

[30] Bojan Jovanovic. "internet of things statistics for 2022 - taking things apart".

[31] Rafiullah Khan, Sarmad Ullah Khan, Rifaqat Zaheer, and Shahid Khan. Future internet: the internet of things architecture, possible applications and key challenges. In *2012 10th international conference on frontiers of information technology*, pages 257–260. IEEE, 2012.

[32] John Kosowatz. 10 smart cities. *Mechanical Engineering*, 142(02):32–37, 2020.

[33] Larry J Kricka. History of disruptions in laboratory medicine: what have we learned from predictions? *Clinical Chemistry and Laboratory Medicine (CCLM)*, 57(3):308–311, 2019.

[34] Ayush Kumar and Teng Joon Lim. A secure contained testbed for analyzing iot botnets. In *International Conference on Testbeds and Research Infrastructures*, pages 124–137. Springer, 2018.

[35] Somansh Kumar and Ashish Jasuja. Air quality monitoring system based on iot using raspberry pi. In *2017 International conference on computing, communication and automation (ICCCA)*, pages 1341–1346. IEEE, 2017.

[36] Adam Laurie and Marcel Holtmann Martin Herfurt. Hacking bluetooth enabled mobile phones and beyond–full disclosure. In *Proceedings of the 21st Chaos Communication Congress, Berliner Congress Center, Berlin, Germany*, pages 27–29, 2004.

[37] Knud Lasse Lueth. "state of the iot 2020: 12 billion iot connections, surpassing non-iot for the first time".

[38] Rwan Mahmoud, Tasneem Yousuf, Fadi Aloul, and Imran Zualkernan. Internet of things (iot) security: Current status, challenges and prospective measures. In *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 336–341, 2015.

[39] Ibrahim Mashal, Osama Alsaryrah, Tein-Yaw Chung, Cheng-Zen Yang, Wen-Hsing Kuo, and Dharma P Agrawal. Choices for interaction with things on internet and underlying issues. *Ad Hoc Networks*, 28:68–90, 2015.

[40] Andrew Meola. Smart farming in 2020: How iot sensors are creating a more efficient precision agriculture industry. *Business Insider*, 2020.

[41] Hichem Mrabet, Sana Belguith, Adeeb Alhomoud, and Abderrazak Jemai. A survey of iot security based on a layered architecture of sensing and data analysis. *Sensors*, 20(13):3625, 2020.

[42] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, Hossein Fereidooni, N Asokan, and Ahmad-Reza Sadeghi. Dïot: A federated self-learning anomaly detection system for iot. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 756–767. IEEE, 2019.

[43] University of New Brunswick(UNB). Nsl-kdd dataset. `https://www.unb.ca/cic/datasets/nsl.html`.

[44] Vamsikrishna Patchava, Hari Babu Kandala, and P Ravi Babu. A smart home automation technique with raspberry pi using iot. In *2015 International conference on smart sensors and systems (IC-SSS)*, pages 1–4. IEEE, 2015.

[45] Eko Sakti Pramukantoro, Widhi Yahya, and Fariz Andri Bakhtiar. Performance evaluation of iot middleware for syntactical interoperability. In *2017 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 29–34. IEEE, 2017.

[46] K Rawlinson. Hp study finds alarming vulnerabilities with internet of things (iot) home security systems. *Hewlett Packard Enterprise*, 2015.

[47] Omar Said and Mehedi Masud. Towards internet of things: Survey and future vision. *International Journal of Computer Networks*, 5(1):1–17, 2013.

[48] Michael Schiefer. Smart home definition and security threats. In *2015 ninth international conference on IT security incident management & IT forensics*, pages 114–118. IEEE, 2015.

[49] Maria Jose Erquiaga Sebastian Garcia, Agustin Parmisano. Iot-23: A labeled dataset with malicious and benign iot network traffic (version 1.0.0) [data set]. `https://www.stratosphereips.org/datasets-iot23`.

[50] Stefano Severi, Francesco Sottile, Giuseppe Abreu, Claudio Pastrone, Maurizio Spirito, and Friedbert Berens. M2m technologies: Enablers for a pervasive internet of things. In *2014 European Conference on Networks and Communications (EuCNC)*, pages 1–5. IEEE, 2014.

[51] Dhvani Shah et al. Iot based biometrics implementation on raspberry pi. *Procedia Computer Science*, 79:328–336, 2016.

[52] Satyajit Sinha. "state of iot 2021: Number of connected iot devices growing 9% to 12.3 billion globally, cellular iot now surpassing 2 billion".

[53] statista. Internet of things (iot) and non-iot active device connections worldwide from 2010 to 2025. `https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/`.

[54] Kehua Su, Jie Li, and Hongbo Fu. Smart city and the applications. In *2011 international conference on electronics, communications and control (ICECC)*, pages 1028–1031. IEEE, 2011.

[55] Priya Suresh, J Vijay Daniel, Velusamy Parthasarathy, and RH Aswathy. A state of the art review on the internet of things (iot) history, technology and fields of deployment. In *2014 International conference on science engineering and management research (ICSEMR)*, pages 1–8. IEEE, 2014.

[56] Lu Tan and Neng Wang. Future internet: The internet of things. In *2010 3rd international conference on advanced computer theory and engineering (ICACTE)*, volume 5, pages V5–376. IEEE, 2010.

[57] Information The UCI KDD Archive and Irvine Computer Science, University of California. Kdd cup 1999 data. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`.

[58] Ann R Thryft. Real-life industrial iot cyberattack scenarios, 2018.

[59] William Turton and Kartikay Mehrotra. Hackers breached colonial pipeline using compromised password. *Bloomberg. Available online at: https://www. bloomberg. com/news/articles/2021-06-04/hackers-breached-colonial-pipeline-using-compromised-password*, 2021.

[60] Ioan Ungurean, Nicoleta-Cristina Gaitan, and Vasile Gheorghita Gaitan. An iot architecture for things from industrial environment. In *2014 10th International Conference on Communications (COMM)*, pages 1–4, 2014.

[61] International Telecommunication Union. "internet of things global standards initiative".

[62] Carnegie Mellon University. “the “only” coke machine on the internet”.

[63] Paul Wang, Amjad Ali, and William Kelly. Data security and threat modeling for smart city infrastructure. In *2015 international conference on cyber security of smart cities, industrial control system and communications (SSIC)*, pages 1–6. IEEE, 2015.

[64] Miao Wu, Ting-Jie Lu, Fei-Yang Ling, Jing Sun, and Hui-Ying Du. Research on the architecture of internet of things. In *2010 3rd international conference on advanced computer theory and engineering (ICACTE)*, volume 5, pages V5–484. IEEE, 2010.

[65] Chen Yang, Weiming Shen, and Xianbin Wang. The internet of things in manufacturing: Key issues and potential applications. *IEEE Systems, Man, and Cybernetics Magazine*, 4(1):6–15, 2018.

[66] Chang-Le Zhong, Zhen Zhu, and Ren-Gen Huang. Study on the iot architecture and gateway technology. In *2015 14th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*, pages 196–199, 2015.

[67] Ivan Zyrianoff, Alexandre Heideker, Dener Silva, João Kleinschmidt, Juha-Pekka Soininen, Tullio Salmon Cinotti, and Carlos Kamienski. Architecting and deploying iot smart applications: A performance–oriented approach. *Sensors*, 20(1):84, 2019.