QATAR UNIVERSITY

COLLEGE OF ENGINEERING

DEEP LEARNING IOT MALWARE DETECTION MODEL FOR IOMT EDGE DEVICES

BY

SULEIMAN KAYED KHARROUB

A Thesis Submitted to

the College of Engineering

in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Computing

January 2021

COMMITTEE PAGE

The members of the Committee approve the Thesis of
Suleiman Kayed Kharroub defended on 18/11/2020.

Dr. Mohsen Guizani
Thesis Supervisor

Dr. Elias Yaacoub
Committee Member

Dr. Khaled Md Khan
Committee Member

Dr. Ala Alfuqaha
Committee Member

Approved:

Dr. Khalid Kamal Naji, Dean, College of Engineering

ABSTRACT

Kharroub, Suleiman, K., Masters : January: 2021, Master of Science in Computing

Title: Deep Learning IoT Malware Detection Model for IoMT Edge Devices

Supervisor of Thesis: Dr. Mohsen Guizani.

Internet of Things (IoT) is defined as the massive collection of physical devices being connected to the Internet. IoT has a positive impact in multiple fields, such as health, agriculture, and power management sectors by advancing them to new technical horizons. However, such advanced technologies introduce security challenges that can negatively affect IoT applications and possibly threaten their existence. In the health sector, for instance, Internet of medical things (IoMT) devices are used to perform tasks such as remote patient monitoring and to gather biometric information. Also, these devices are used as a base for several healthcare procedures such as prescribing medication. Several security breaches can occur to IoMT devices that may expose human privacy and security since the data collected and processed is very sensitive. In this thesis, we provide a light-weight malware detection deep learning model. The model is deployed on IoMT edge devices that can detect IoT specific malware. The proposed models utilize gray-scale images produced by the binary of malware files to classify malware from goodwares. The achieved results were promising in terms of malware classification accuracy, which might help prevent malware and secure the dedicated systems for IoMT devices and applications.

DEDICATION

For my parents, my family, my friends, and for a better society.

ACKNOWLEDGMENTS

I would like to acknowledge my colleagues that helped me in coding the proposed solution, my friends who helped in creating figures to clarify ideas, and most importantly my advisors for directing me on the right path and being patient with me.

TABLE OF CONTENTS

DEDICATION iv

ACKNOWLEDGMENTS v

LIST OF TABLES vii

LIST OF FIGURES viii

Chapter 1: Introduction 1

Chapter 2: Literature Review 3

Chapter Overview 3

Introduction 3

Literature Review 4

Centralized: 5

Non-Centralized 8

Low-Weight Security 13

Chapter Conclusion 15

Chapter 3: Base Model 17

Chapter Overview 17

Briefing 17

Related Work 18

Methodology 19

Simulation Process 21

Results 23

Chapter Conclusion 25

Chapter 4: Improved Lightweight Model 27
Chapter Overview 27
Briefing 27
Motivation 28
Methodology 29
Acquiring Data 30
Data Pre-Processing 30
Feature Extraction 31
Classification 33
Tensorflow Lite 33
Simulation Set-up Specifications 34
Results 37
Tensorflow Model 37
Tensorflow Lite Model 39
Comparison 40
Chapter Conclusion 42
Chapter 5: Conclusion and Future Work 44
Future Work 44
References 46

LIST OF TABLES

Table 4.1. Tensorflow F1-Score, Recall, and Precision 38

Table 4.2. Tensorflow Confusion Matrix.. 39

Table 4.3. Tensorflow-Lite F1-Score, Recall, and Precision 39

Table 4.4. Tensorflow-Lite Confusion Matrix 40

Table 4.5. Deep Learning Model Results [28] 41

Table 4.6. Deep Learning Performance for Android IoT devices [29] 41

LIST OF FIGURES

Figure 2.1. The IoMT Landscape [5] 5

Figure 2.2. Example of a typical encryption and decryption scheme 8

Figure 2.3. Cryptographic process using secure transport layer protocol [10].......... 10

Figure 2.4. Blockchain Structure 11

Figure 3.1. Example of Produced Grayscale Image 20

Figure 3.2. Example of Produced Hilbert Curve Image 20

Figure 3.3. CNN Model Summary 21

Figure 3.4. Comparison Tables of Related Scores and Metrics 24

Figure 3.5. Accuracy VS Size of Training Set 25

Figure 4.1. Proposed IoMT Architecture 29

Figure 4.2. Example ReLU Activation Output Graph [26] 32

Figure 4.3. Hardware Specification Used to Train the Model 35

Figure 4.4. Tensorflow Model Accuracy Histogram on PC Node 38

CHAPTER 1: INTRODUCTION

In this thesis, we tackled the security challenge of malware detection for the Internet of Things (IoT). During my studies, I was heavily involved in the security aspect of applications, systems, and networks. Due to the lack of security standards for IoT devices and applications, achieving security became a major challenge and a hot research topic. The processing power and energy limitations of an IoT device make it even harder to propose a security mechanism that is viable to use in the industry. This is especially important when we involve IoT in the medical field creating what is referred to by the Internet of Medical Things (IoMT). IoMT enabled easier access to healthcare and advanced the medical field in multiple ways giving birth to technologies like remote surgeries, and remote patient monitoring. Securing such applications is a crucial task due to the extreme sensitivity of the collected and processed data. IoMT applications and devices are directly involved with the well-being of humans which makes failures in such systems fatal. With that being said, it became my goal to propose a solution that mitigates one of the major challenges in IoMT security; IoT malware detection. During my graduate studies, I dedicated my research towards the goal of IoT and IoMT security achieving multiple publications in that field. In this thesis, I will be showcasing all of the steps that I took to achieve IoT and IoMT malware detection through the use of machine learning and deep learning. The main problem that we are trying to solve is the ability to identify IoT-specific malware on a network level. This thesis will be segmented into the following chapters:

- Chapter 2: Showcases a survey paper that we published in IWCMC 2020 about different encryption and security techniques in IoMT.

- Chapter 3: Demonstrates our first attempt at creating a deep learning model that can classify IoT specific malware using grayscale images created from the binary of files.
- Chapter 4: Expands on our previous work in Chapter 3, converting the created model into a lightweight version that can run directly on IoT devices.
- Chapter 5: Concludes this thesis with remarks and future work to be done.

CHAPTER 2: LITERATURE REVIEW

Chapter Overview

In this chapter, we summarize the work done in our published survey paper [1]. This survey paper was our first step into understanding the literature regarding IoT and IoMT security techniques and challenges making it an important cornerstone for this thesis. Understating the state-of-the-art in IoT/IoMT security is a crucial step to develop a solution that attempts to solve the problem at hand and improve the literature. This chapter summarizes multiple IoT security approaches each with its benefits and limitations to understand each approach and the reasons behind why we chose machine learning to be our method of choice for IoT/IoMT malware detection.

Introduction

As of late, it became a simple matter to get healthcare no matter the place due to technology. While said technology cannot simply cure chronic diseases, it can increase healthcare accessibility all over the world. Recent advancements in the Internet of Things (IoT) reached very sophisticated levels of precision and reliability to the point where we can safely integrate IoT devices and their applications in our daily lives. Such technology allowed humanity to have smart homes, smart cars, smart electric grids, and many more applications that automate trivial and non-trivial tasks [2]. Most importantly, IoT devices are now being used in the medical field for various applications such as gathering biometric information from patients automatically over time intervals, remotely give patients their medicine dosage depending on certain readings, and doing remote surgeries where the doctor might be in a different country than the patient but is still able to control his/her tools and operate as if the patient is right next to him/her. All of those technologies made the healthcare sector very advanced and made it much easier for people all over the world to get the medical care appropriate for them at the right time [3]. However, as we depend more and more on IoMT, the risk on human lives increase. IoT devices are indeed somewhat reliable and therefore we can safely implement them in a controlled environment. But that is not always the case with IoMT and the healthcare section. As we mentioned before, most of the medical applications of IoT require connection over the cloud to operate as intended. And when communication over the cloud is mandatory, multiple security issues arise. Risks like exposing patient data, data eavesdropping, unauthorized data access, location privacy, and medical hijacking are some of the security issues in IoMT systems [4]. Security plays a major role in making sure IoMT applications are working as intended without risking human lives in the process. The CIA triad (Confidentiality, Integrity, Availability) should be kept in mind when developing medical applications and the way those applications communicate with other applications, systems, devices, and through the cloud. Unfortunately, IoT manufacturers do not agree on specific security standards when making IoT devices. Which makes it remarkably harder to come up with security solutions due to compatibility issues.

Literature Review

In this section, we will provide a comprehensive literature review for different IoMT encryption methods and categorize them under three categories, centralized, non-centralized, and low weight security techniques. An important thing to mention is that IoT devices are very limited in terms of processing power and energy making it difficult to utilize the standard security and encryption techniques that are usually used in more powerful devices like PCs and Servers. However, some of the techniques that we will mention utilize powerful devices such as a server to handle part of the security process.

Centralized:

A centralized approach is an approach where a powerful device is used to perform process-heavy tasks, pushing the workload away from IoMT devices. In these types of solutions, the goal is usually to use classic encryption, decryption, and key sharing techniques within an IoT environment. Since IoT devices are limited in processing power and energy, so typical security techniques cannot work if we deploy them on the IoT device itself. However, processing the complicated security techniques on a more powerful node (Centralized Node) like a server or a computer is one solution to this issue [2] [3].

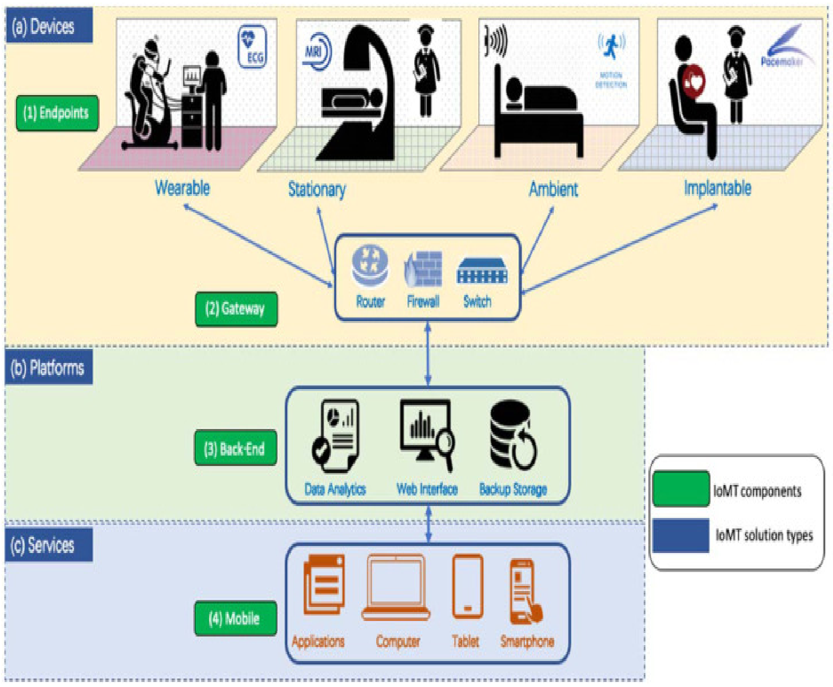


Figure 2.1: The IoMT Landscape [5]

In [5] a novel ontological scenario-based methodology is used to develop a web-based IoMT Security Assessment Framework (IoMT-SAF). The proposed system is used to recommend security features and evaluate security and limitation in IoMT solution. IoMT-SAF supports the selection of a solution that matches the stakeholder's security objectives and supports the decision-making process. The value of IoMT-SAF lies in its extensibility, attention to detail, as well as customizability for different stakeholders while keeping top performance according to technology and medical standards. Several components defined by the Open Web Application Security Project (OWASP) are presented in the paper [5] as the classic components in IoMT solutions such as endpoints, where medical devices are connected to hospital networks, the Internet, or to other medical devices. From the above Figure 2.1 which illustrates the layout of IoMT Landscape and is presented as the following:

1. Represents the endpoints section, The DFA showcases connected medical devices (IoMT endpoints) as medical devices that are connected to hospital networks, remote cloud, other devices, or the internet. The work done in [5] consider non-medical devices that can be used in IoMT environments such as ambient sensors.

2. Represents gateways, they act as a support mechanism for enhancing the connectivity of weak endpoints by working as a bridge.

3. Back-end section, current back-end servers are used by IoT systems to run IoMT solutions, process data, And store data produced from the network.

4. Mobile Devices/Applications that are usually deployed in IoT systems to grant remote control access for endpoints and back-end management.

However, essential components in an IoMT solution are different based on the used framework that we are working with. IoMT devices are classified as follows:

- Wearable devices (e.g., heart monitors).
- Implantable devices (e.g., embedded cardiac function monitors).
- Ambient (e.g., door sensors).
- Stationary (e.g., computerized tomography scanners).

The nature of the IoMT platforms also vary, but the most common ones are cloud-based platforms that are used to establish smart IoT devices and IoT applications. Such platforms offer administrators centralized back-end management capabilities like analytics backup through web interfaces, and ecosystem administration reports [6]. Services are used to provide edge computing to analyze the collected data and to integrate IoMT with other systems [7]. Typically, IoMT solutions are a combination of systems, some applications utilize the accounting and long-range monitoring systems to come up with a solution. Therefore, securing IoMT applications depend on what systems are involved in a given solution and that is why it is typically difficult to come up with a standardized general solution. Following the centralized approach, some solutions already exist that support symmetric secret key -or shared key- in IoT. In such approaches, the IoT devices initiate mutual communication between the sensors and base station to agree on a common key (shared-key), another connection between a base station and key distribution center is established after that. Since IoT devices are limited in terms of power, such approaches prove to be difficult to implement since regular encryption and decryption techniques like RSA and ECC cannot be used within IoT environments [6][7].

Non-Centralized

The Non-Centralized approach came about to mitigate the limitations of the centralized approach. Since IoT devices are limited in terms of resources, centralized approaches tend to be very difficult to implement. On the other hand, non-centralized approaches tend to have different solutions to reduce the load on IoT devices and use the nature of their Ad-hoc connectivity. When an IoT device communicates with the cloud, the security risks increase dramatically. Although having an infrastructure can help us detect security breaches and better assess the overall security of the network, we will learn in this section that it is not always the case.

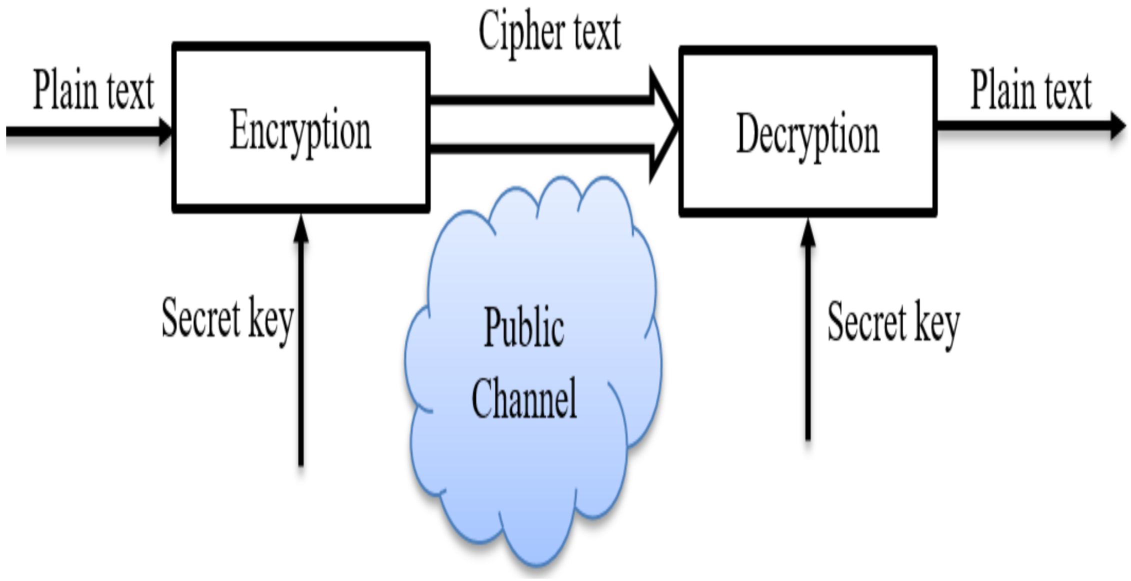


Figure 2.2: Example of a typical encryption and decryption scheme

Figure 2.2 shows the message as plaintext, also identified as the original message, Which is changed to a ciphertext through an encryption algorithm. Each ciphertext goes through the mentioned public channel whenever a message is sent from sender to receiver. When the receiver gets the message, it gets decrypted to plaintext. As mentioned in [8], we can implement data encryption at three communication levels: node level encryption, link-level encryption, and end-to-end encryption. In the case of intermediate nodes in a given link, each received message from the former link will first be decrypted to plaintext, then, using the secret key of the next link, the plaintext is encrypted into ciphertext. On the other hand, node encryption does not allow any plaintext messages to be formed within network nodes. That is why node encryption can be a good solution to provide high security for network data. Lastly, in end-to-end encryption, messages are not decrypted until they are fully transmitted to the destination [9]. In that case, the messages in a given transmission are always encrypted as ciphertext, minimizing the risk of information leakage if a node gets corrupted. To secure healthcare connections, key management protocols play a major role in the overall security process. The use of composite encryption algorithms or transmission protocols can heavily affect transmission rate making it impossible to transmit data in some cases. Moreover, striking a balance between energy consumption and acceptable security measurements is a problem that needs to be solved methodically. To control malicious activities, the transmission protocol of data requires a secure channel between IoMT devices and the cloud. The research work in [10] proposed PROTeCt—Privacy architecture which is a cloud-based security approach for IoT. The model enhances user privacy by enforcing privacy at the device level instead of the gateway. In that case, we eliminate the issue of single point of failure which increases the overall security and fault tolerance of a given system. The results show that the proposed solution is promising in terms of increasing the privacy and security of IoT devices.

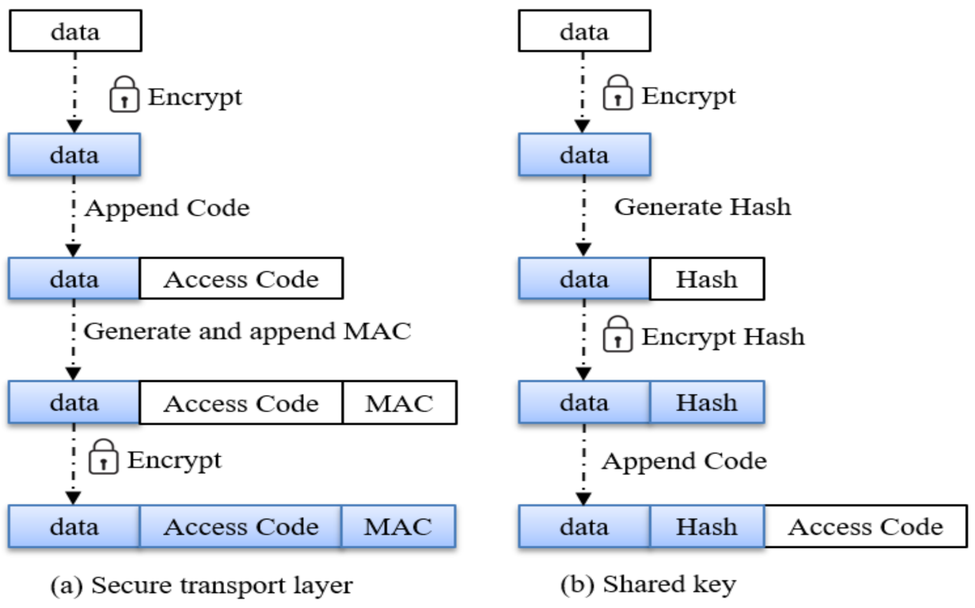


Figure 2.3: Cryptographic process using secure transport layer protocol [10]

Therefore, every single transformation from IoMT to the cloud should include a code; it recognizes the device and proves it has permission to store the data on behalf of the user. The following Figure 2.3 shows that the cryptographic operations and message format of transmission while using a secure transport layer protocol. It also highlights two important processes, (a) or a shared key with the cloud provider (b). The blue and white boxes represent encrypted and plain content, respectively. The data is generated through IoMT devices and encrypted two times, the first one at the application layer, and the transport layer is responsible for the second encryption. Thus, the entire message is encrypted, which includes the encrypted data, MAC and access code [10]. Another interesting approach that is completely non- centralized is using block-chains to establish security within an IoT network. Blockchains proved to be useful in multiple IoT related technologies, especially in the field of security [11]. It provides an immutable ledger where data can be stored without the fear of it being changed unexpectedly later on. This ledger can be accessed by the devices connected to the network so that every device can verify and consent to any updates or changes in the ledger. Thus, making it very suitable for security applications like key sharing.

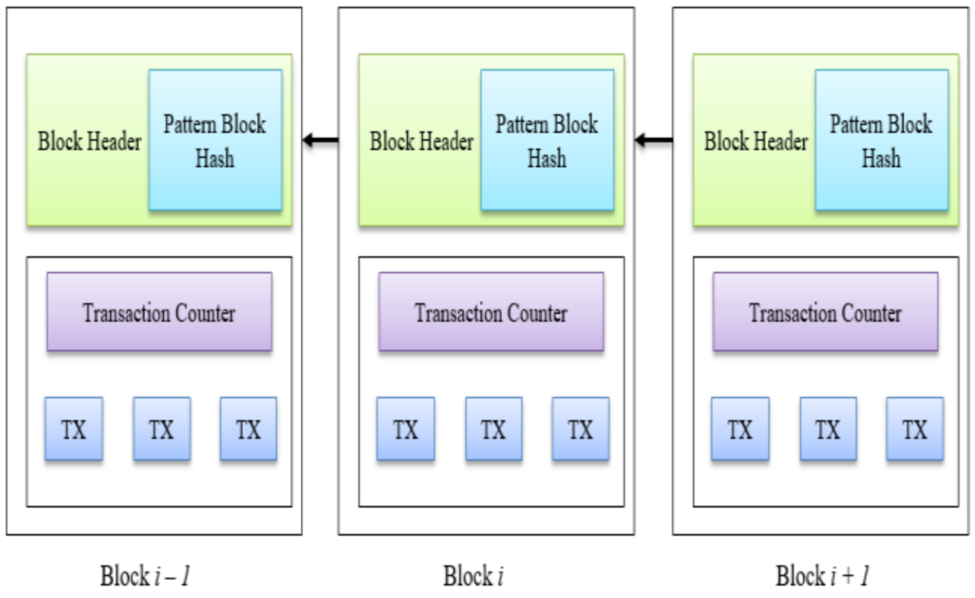


Figure 2.4: Blockchain Structure

Every blockchain ledger contains a series of blocks as represented in Figure 2.4. Within every block, we have data and a hash that points to the previous block in the series. Before a block is established and added to the ledger, every device on the network must verify its legitimacy. Also, if we want to change the data in a specific block, this action must be verified for legitimacy by every device on the network before changing the data in one block. One noticeable drawback of the blockchain is the fact that if we change the data of one block, we must change the hash pointers of every block that comes after it. Which is a costly process in terms of energy and processing power. In [12] they proposed a solution for smart home IoT security that implements blockchains. The solution proposed is based on the following components:

Transactions

It is defined as the mutual communication between local devices and-or overlay nodes. Different transactions correspond to different functions within the smart home system.

- "Store" transaction is created by devices to store data.
- "Access" transaction is generated by a service provider (SP) or the homeowner to access the cloud storage.
- "Monitor" transaction is generated by the homeowner or SPs to periodically monitor device information.
- "Genesis" transaction is to add a new device to the network.
- "Remove" transaction is to remove a device from the network.

Transactions mentioned in the list above, utilize a shared key to secure communication. Light-weight-hashing is used to detect any change in a specific transaction content during transmission. All out-going and in-going transactions of the smart home are stored in a local private BlockChain.

Local Block-Chain (BC)

Within every smart home, there is a local private BC that maintains a record of all transactions that occur in the home network. After the "Genesis" transaction, each consecutive transaction is linked with another transaction making the immutable ledger. Each block contains two headers, the block header, and the policy header. Policy header utilized for authorizing connected devices, and enforcing policies defined by the homeowner. Every block header includes the hash of the previous block to keep the integrity of the BC.

Home Miner

A device that is responsible for processing incoming and outgoing transactions of the network. It can be integrated within a network gateway or as a separate device. Like current central security devices, the home miner authenticates, audits, and authorizes transactions in the network. It also generates "Genisis" transactions and handles key distribution and key updates. It is also responsible for collecting all transactions into a block to append in the BC. It also handles local storage, which provides adds to the maximum network capacity [12].

Low-Weight Security

Some research efforts are done in the area of asymmetric key sharing between IoT devices without the need for a base station or infrastructure. In [13] they propose a solution that divides the resource-limited sensors under the same environment presented by the Light-Weight Implementation Guidance (LWIG) working group into two different classes; Class 0 super-light sensors and Class 1 or above sensors. They deploy the authentication method and session key sharing method according to the performance of each group of sensors. They have the following parameters for their solution:

- I is the authentication initiator.
- R is the responder to the message from I.

- R1 and R2 are random nonce values.
- IR is the safely shared key in advance between I and R.
- Sk is the session key shared between I and R.

The proposed system allows devices to authenticate each other and share a key to be used for symmetric encryption. In their solution, it is assumed that the secret key value and the encryption function $E_k(x)$ are safely saved in each device at the setup stage. Then the procedure of the mutual M2M authentication is as follows:

1. $I \rightarrow R : \{I_{ID}, R1\}$ I generate a random number R1, and sends it with its ID, I_{ID} to R.

2. $R \rightarrow I : \{R_{ID}, E_{KIR}(R1||R2)\}$ R encrypts the received R1 and then, the coherently generated R2 with kIR and sends those values to I along with its ID (R_{ID}).

3. I and R XOR-operate R1 and R2, and encrypt the value with to generate the session key SK.

4. $I \rightarrow R : \{E_{SK}(R2)\}$ I encrypt R2 with the session key SK, then re-transmits it to R to get the authentication.

   The following is a continuation of the communication process between the IoT devices after the shared key has been established:

5. $R \rightarrow I : \{E_{SK}(DATA1)\}$ Sends back the authentication to I from the previous step (4). Then encrypts the data that it wants to transmit using SK before transmitting. Lastly, it sends the encrypted data to I.

6. $I \rightarrow R : \{E_{SK}(DATA2)\}$ Similarly, I then use SK to encrypt the data that it wants to send.

7. Session end. After data has been communicated, the session ends and both devices dispose of the shared key SK.

In this approach, every IoT device can authenticate other devices and share a new key with every communication session. Although this type of security is not fully robust against the toughest of security attacks. But it provides a sufficient level of protection due to the fast and rapid nature of data exchange between IoT devices it becomes difficult for attackers to figure out the shared key (SK) in time before it is changed again. With this solution, IoMT devices that are present within the same network (i.e. deployed on the same patient) can communicate securely with each other. Maintaining secure enough communication to safely ensure the integrity and confidentiality of vital health data.

Chapter Conclusion

After discussing different state-of-the-art techniques in IoT security. We concluded that such techniques are not sufficient to satisfy the security requirements of IoT devices and systems. A typical IoT ecosystem consists of two important types of devices; a sensor that collects data from the physical world, and an actuator that alters physical world attributes based on signals that it receives from other sensors or commands issued by users. Actuator functionalities can vary in terms of criticalness, a simple example could be turning on the light, a more complicated example could be insulin injection pumps. While failure in simplistic actuator functionalities is tolerable, failure in the more complicated ones can directly risk human lives. Therefore, we need an adaptive approach if we want to implement an acceptable level of security. Adaptive technologies like machine learning and deep learning can be very beneficial in this case. We can train models -using known IoT malware- that can classify any given data as malware or goodware. Machine learning is especially useful here because it can also detect unknown malware based on what it learned from known malware in terms of patterns and features. Then the idea came to mind to implement such an approach which we discuss in the following chapter.

CHAPTER 3: BASE MODEL

Chapter Overview

This chapter explains the work that we published in [14]. After discussing the literature in the previous chapter, we realized that machine learning can be implemented for IoT security purposes. Due to its adaptive nature, machine learning proves to be useful in implementing malware detection. Therefore, in this chapter, we explain how we created a lightweight machine learning model that can classify data accurately into IoT specific malware or goodware.

Briefing

Efforts in the research field have been made to create a standard for IoT security that covers all IoT fields. An example of such efforts can be seen in ISO/IEC 27030 and ISO/IEC 30141. which are used as tools to assess an IoT framework by looking into the trustworthiness to evaluate the reliability of an IoT system and how safe is it taking into consideration the context of said system [15]. Governments in the US and EU are looking forward to having a standard for IoT security that is proved to be robust. However, there are two main concerns regarding the development of such standard; the first one being the diversity of IoT applications and their suggested standards make it challenging to find common ground that can unify them, the second one is mainly due to lack of information regarding implementations and review rates of suggested standards which makes it difficult to assess each standard with its respective context to signify its impact. Even though there are bunches of endeavors in the IoT security field, it stays dubious what is the best innovation to use to provide acceptable security without negatively affecting energy utilization. Since IoT gadgets are normally extremely restricted in terms of energy and processing power, so it is challenging to propose a lightweight, high-security solution. In this chapter, we are targeting network-level packet filtering to achieve IoT malware detection based on Convolutional Neural Networks (CNN). With this, we will be able to recognize malicious nodes currently connected to a network and safely apply the appropriate security mitigation.

Related Work

This section mentions recent advancements and approaches that use different technologies to fulfill the required aspects in safe and secure IoT systems. In [16], a blockchain-based control scheme is used to fulfill IoT security requirements. The proposed solution utilizes the fact that blockchain has an immutable ledger that resists un-authorized data altering. Blockchains are a robust security solution, but they require high processing power as mentioned previously in Chapter 2. In the case of IoT, implementing blockchain can heavily negatively affect energy consumption, making blockchains an invalid option. [17] proposes a different security approach specifically for Android-based IoT device. By utilizing a fusion of machine learning and blockchains to establish malware detection. The proposed methodology utilizes machine learning to extract malware information, classify malware, then write that information to the blockchain making the information visible to all nodes in the network. The main issues with the two previously mentioned papers [16][17] is the high processing power required to implement such solutions. In this case, security solutions that require heavy processing prove to be counterproductive to the overall performance of the network. This means that such solutions are not a viable option in our case. Therefore, we need to look at lightweight security approaches that do not affect our network negatively. In [18], the authors proposed a lightweight malware classification model that utilize images to classify IoT specific malware. The fact that the proposed model is lightweight, makes it a much more viable option than the previously mentioned solutions in the case of IoT. The authors mentioned that DDos malware mitigation in an IoT environment was their area of focus in that work. By extracting features from grayscale images created from the malware binary, they were able to train a lightweight CNN model that can classify malware based on their corresponding grayscale image. The proposed model achieved a classification accuracy of 94% when testing using a batch of DDoS malware and goodware. The used dataset in the proposed work can be found in [19], where they mainly focused on IoT malware collecting that can be used for research purposes.

Methodology

Our approach incorporates taking a gander at specific advances that will hugely improve the result of a classifier, particularly when utilizing CNN as the principal model for arrangement. The proposed solution should incorporate a lightweight CNN model which should deal with various situations utilizing distinctive data representations that we will be using to test and evaluate the proposed model. We were able to acquire a dataset from IoTPoT [19], where he wrote a paper explaining the assembling of the dataset. The dataset consists of 5000 samples of IoT specific malware that we used to train and test the model. However, the dataset was not labeled. Therefore we had to use a labeling service called VirusTotal API [20] to label all entries in our dataset. The majority of the dataset contains malware from Linux.Mirai and Linux.Gafgyt malware families. The dataset was comprehensive of extremely light malware which can only spread through IoT devices, which restricted the variety of obtained malicious binary files. This will make classification simpler and convenient as we will be considering the two mentioned malware families as the output categories. We also collected a sample of 1000 goodwares that consist of Linux based software since usually IoT devices use a Unix based kernel. We added the goodware samples to the malware samples to accumulate our finalized dataset. In terms of data representation, we decided to include two forms of images to represent the binary files that we have in our dataset. The first representation includes reading the binary files in bytes then convert each byte into a value between 0 and 255 to produce a grayscale image seen in Figure 3.1.

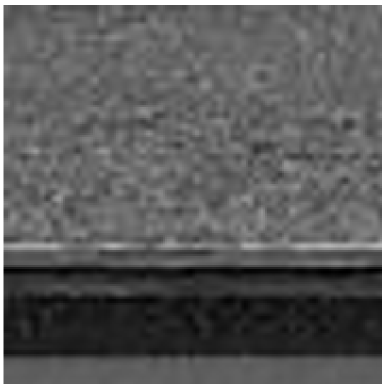


Figure 3.1: Example of Produced Grayscale Image

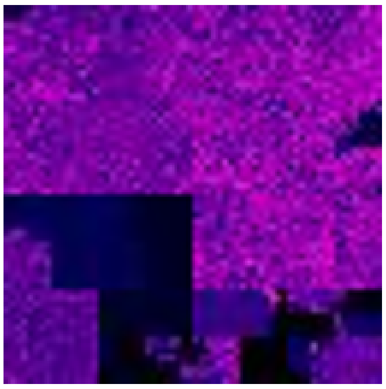
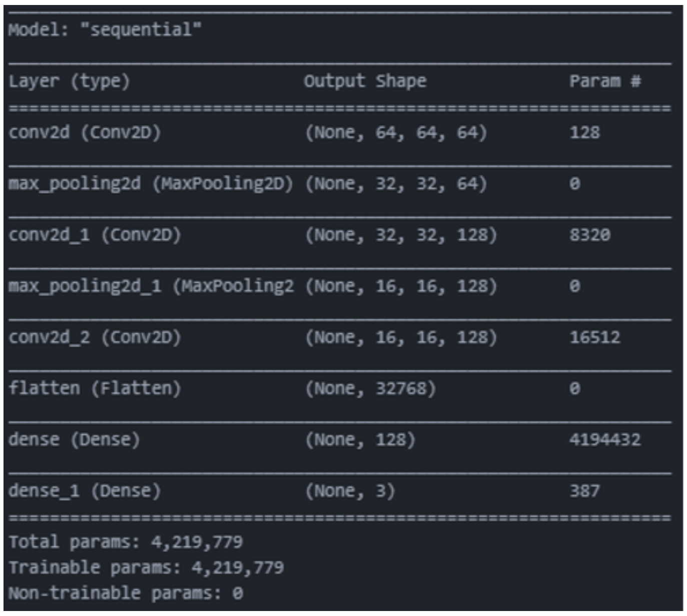


Figure 3.2: Example of Produced Hilbert Curve Image

The second representation of binary files is using Hilbert curve, which is an entropy cluster image that is produced by reading the binary files in bytes then passing each byte through an image generating function used from [21]. An example of the produced Hilbert curve image can be seen in Figure 3.2.



```
_________________________________________________________________
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 64, 64, 64)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 128)       8320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 128)       16512
_________________________________________________________________
flatten (Flatten)            (None, 32768)             0
_________________________________________________________________
dense (Dense)                (None, 128)               4194432
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 387
=================================================================
Total params: 4,219,779
Trainable params: 4,219,779
Non-trainable params: 0
```

Figure 3.3: CNN Model Summary

Figure 3.3 showcase the summary of our proposed CNN model. The main highlight in this is the fact that we added an extra convolution layer which will help in increasing the overall accuracy. Another point worth mentioning is that we expand on the features that we have rather than reducing them. Given that we have a limited size of inputs, we had to expand on our current features to give a better classification process. As for the gate function, we used ReLu which is considered reliable in most cases. In the case of other training metrics like the image_size, epochs, etc. they were all selected according to our own experiments through trial and error to provide the highest possible accuracy.

Simulation Process

To train the model, we used the following hardware components:

CPU Architecture and OS: Windows 10 Pro with Intel i7-7700HQ – 8GB RAM GPU Architecture: NVIDIA GeForce GTX 1050 Max-Q (2GB RAM)

And we used the following software frameworks:

- TensorFlow => 2.0+
- PIL
- matplotlib.pyplot
- Python => 3.7+

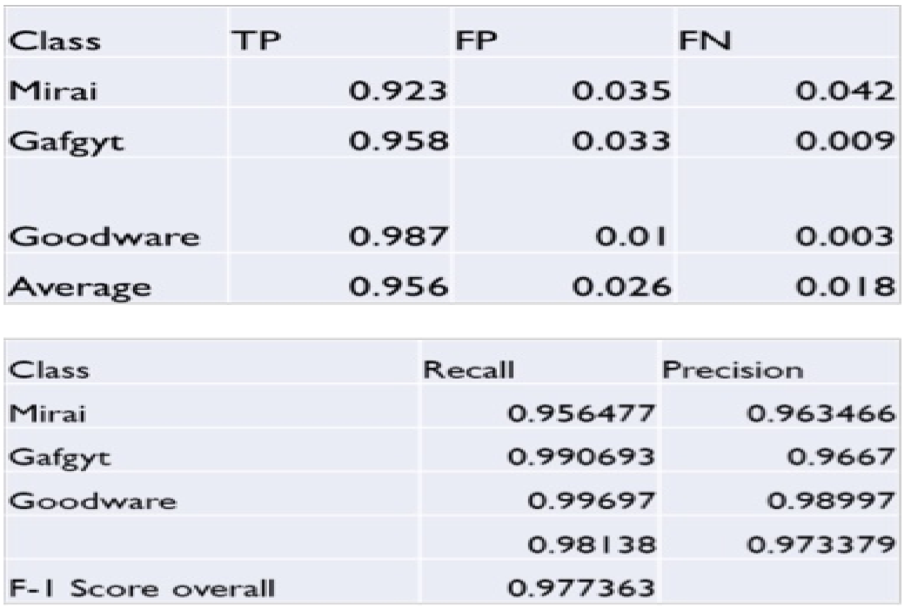
We will be conducting our experiment following three simple steps which are; data initialization, model training, model testing. And we will be repeating this process until producing good results. As mentioned previously we will be categorizing our outputs into three classes; Gafgyt and Mirai being the malware families, and goodware representing benign software. We also used 64x64 images since it provides the highest efficiency without compromising model accuracy during training. Our main goal here is to create a balance in terms of model performance and viability of use by optimizing as many input parameters as we can. As for the number of iterations, referred to by epochs. One epoch is defined to be one passing of the entire dataset forward and backward through the neural network according to [22] making 1 epoch a huge investment. Since our main worry here is the viability of use, we deconstructed each epoch into multiple smaller training batches. Each batch in Tensorflow consists of 32 entries by default. Therefore, after testing our model using a different number of epochs, the accuracy curve flattens after five epochs. In that case, if we train the model for more than five epochs it will harm classification accuracy. To save time and energy we decided to use only five epochs to train our model with a batch size of 32 since batch size doesn't affect training time. In terms of loss value optimization, we attempt to optimize the value of 'sparse_categorical_crossentropy' alongside 'adam' as our targeted optimizer function. A machine learning algorithm utilizes loss functions to learn, it is also used to evaluate the performance of a model on how well it can predict the classification of given data. If a prediction is very far off from the true value, the loss function will give us a large number, and if a prediction is close to the true value, the loss function will give us a small number. In our case, we used the default parameters for the used functions to make sure that we do not overfit our model.

Results

After we set up the data and the model, the experimental parameters were as follows:

- Goodware files: 1000
- IoT malware files: 4000
- Number of epochs: 5
- Convolution layers: 3
- Pooling layers: 3
- Input image size and channel: (64, 64, 1)
- Input data are randomized

We assume that we captured the packets or binary files that contain malicious code. Afterward, we run our pre-processing script to produce the grayscale images from the binary of the captured data. This assumption will help in understating how the model can be used in the future. After training the model on several trials, we grasped a rough estimation of how much data we need for training and how much we need for testing. The split we opted for was as follows; 70% of our dataset would be used for training, and the remaining 30% will be used for testing and evaluation purposes. Afterward, we produced key evaluation values for any machine learning model which includes Precision, Recall, and F-1 Scores of our proposed model. Figure 3.4 shows a comparison between different scores compared to solutions in the literature, in [18] to be specific. Our proposed model provided much higher accuracy and fewer misclassifications.



Class	TP	FP	FN
Mirai	0.923	0.035	0.042
Gafgyt	0.958	0.033	0.009
Goodware	0.987	0.01	0.003
Average	0.956	0.026	0.018

Class	Recall	Precision
Mirai	0.956477	0.963466
Gafgyt	0.990693	0.9667
Goodware	0.99697	0.98997
	0.98138	0.973379
F-1 Score overall	0.977363	

Figure 3.4: Comparison Tables of Related Scores and Metrics

As we can see from the overall F1-scores, the small modifications that we showcased in this chapter proved to be very beneficial on the overall evaluation metrics. And most importantly, our model maintained the balance between performance and being lightweight due to the low number of layers in it. To showcase the difference between using the hilbert curve image compared to the grayscale image for classification purposes we present our results in Figure 3.5. It is worth noting that the accuracy is very similar between the two input data, even though the Hilbert Curve image is an RGB image with 3 channels. We also notice that the training time is not different between the two input data representation, that is why we did not include it in this simulation.

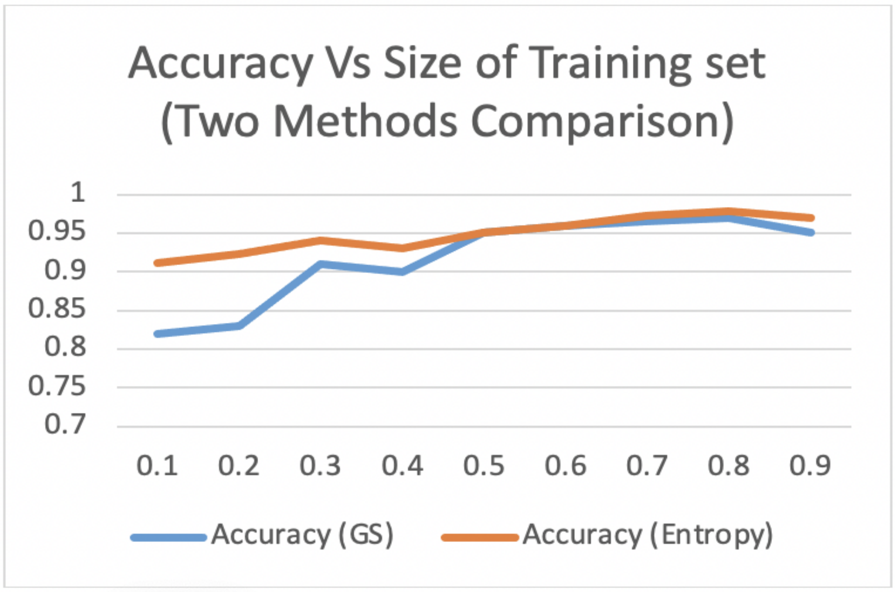


Figure 3.5: Accuracy VS Size of Training Set

The main focus of this chapter is to create an optimized CNN model that can help in malware detection specifically in IoT. Given a relatively small HDF5 file, the proposed model can identify whether or not a given file or packet is considered malicious or benign. According to [23], HDF5 files have very minimal limits, which means that it can be used in multiple devices assuming that we provide the appropriate user interface (UI).

Chapter Conclusion

To conclude this chapter, we were able to create a lightweight CNN model that can accurately classify IoT-specific malware. The results that we found were really impressive in terms of IoT specific malware detection compared to the literature as demonstrated. We believe that deploying this model over IoT devices is possible but we did not attempt to do that in this chapter. We will be exploring the deployment of the proposed model over IoT devices in the next chapter.

CHAPTER 4: IMPROVED LIGHTWEIGHT MODEL

Chapter Overview

This chapter expands on the work done in the previous chapter (Chapter 3) by taking the previously proposed model and convert it into an even lighter model that can run directly and smoothly on IoT and IoMT devices. Pushing the malware detection process to edge devices will make it faster and more energy-efficient. Since we eliminate the need to communicate with an external server for malware detection purposes, this makes it so that each IoT device can verify data that it receives individually. The process of creating such a lightweight deep learning model and the methodology behind it is recorded in this chapter.

Briefing

As we mentioned, the implications of a security breach on IoMT devices and applications are quite dangerous on human lives where we cannot tolerate any kind of failure. Therefore, a security mechanism is required to mitigate such implications. Recently, the use of deep learning in security applications is receiving a lot of attention in the research field. Due to the ability of deep learning techniques to adapt to new or unknown malware and malicious code, they prove to be extremely useful in the security department [24]. However, training a deep learning model is generally a heavy computational process that usually is not applicable to be used over IoT and IoMT devices. But there are ways that we can benefit from the security implications of a deep learning model and deploy it over IoMT/IoT devices with as low demand in terms of processing power. In this chapter, we will propose a lightweight deep learning convolution neural networks

(CNN) model that can detect IoT specific malware on an edge devices level. We will deploy this lightweight model on IoMT devices which will help in securing peer-to-peer communications as well as cloud communications.

Motivation

The main goal of this chapter is to propose a secure method for IoMT edge devices to exchange and process data in a secure manner. Because IoMT and IoT are significantly prone to security attacks and breaches due to the lack of security measurements in several stages such as during device manufacturing, communication protocols, and software. To mitigate this challenge, we develop a CNN deep learning model that is trained to detect IoT specific malware using grayscale images created via the binary code of files received by the IoMT device. After training the model, we will convert it into a lighter version to be deployed over IoMT devices. We believe that this solution provides a sufficient level of security for IoMT devices on the edge level. The proposed architecture of the IoMT system is shown in Figure 4.1 where data passes through every IoT device gets transformed into grayscale images and then fed to the proposed malware detection model before further processing. Since the model is deployed on an edge level, it can also secure peer-to-peer communication between IoMT devices.

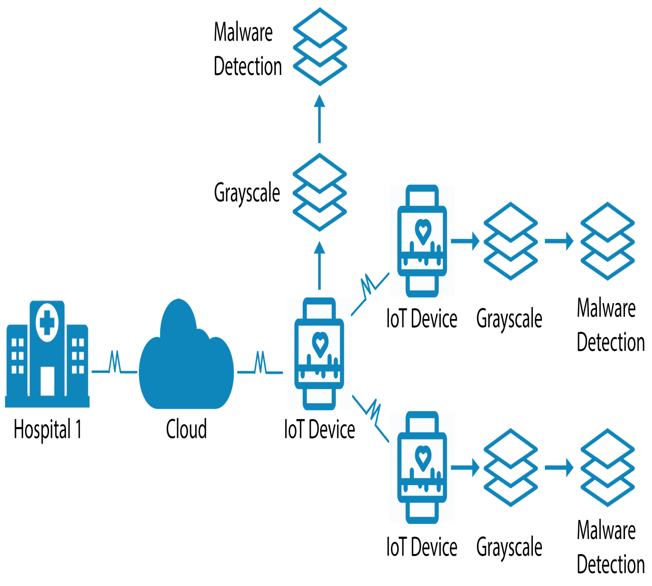


Figure 4.1: Proposed IoMT Architecture

Pushing the malware detection process to edge devices will improve malware detection accuracy, removes unnecessary overhead, and reduce detection time which is vital to IoMT applications. We believe that given the proposed architecture, the overall security accuracy of an IoMT system would improve.

Methodology

This section will cover all the details and implementation process that we went through to achieve the proposed solution. As discussed before, the proposed solution in this chapter is an extension of our work in the previous publication [14] mentioned in Chapter 3. We discuss the main building blocks of the model, what dataset is used, how we pre-processed the data, what percentages from the dataset we used for training and testing, and how we exported the model to run on an IoT edge device. The main purpose of the work in this chapter is to convert the model that we created in [14] into a lightweight model that can run over less powerful devices such as IoMT and IoT devices while maintaining acceptable levels of malware classifications accuracy. There are specific steps that need to be followed in the development of any deep learning or machine learning models:

Acquiring Data

Before starting to explain the deep learning model, we first have to acquire an IoT Malware dataset. The dataset used in this chapter includes the binary files of the IoT specific malware that was collected by Yoshioka from IoTPoT, where he wrote a research paper explaining the assembly of this dataset in [25]. The mentioned dataset initially is not labeled, therefore we used an API called VirusTotal to label all the data that we acquired [20]. After a deep inspection of the dataset, we found that all data falls into two categories, Linux Gafgyt or Linux Mirai malware family. In the proposed solution, we will consider those two families as the categories for the classifier. We also included around 1000 samples of Linux based software to be considered and labeled as goodware making our classification categories as:

$$Categories = ['Gafgyt','Mirai','Goodware'](1)$$

Data Pre-Processing

Before using the data that we acquired and labeled, we first have to unify and normalize data into a specific representation. In our previous work [14], we represented the data in two forms. The first one being grayscale image representation and the other one is as Hilbert curve (Entropy) image representation as previously mentioned in the methodology section of Chapter 3. However, in this chapter, we will be only focusing on the grayscale image representation since it requires less processing power to produce. Given that we will be deploying the model over IoT devices, the lower the processing power required, the more usable our model will be. We also use a standard size for the grayscale images and that is 64x64, which we believe is enough to achieve our goal of malware detection.

Feature Extraction

After pre-processing the data, we need to extract specific features from the grayscale images to correctly classify malware from goodware. We first need to build our CNN model which we already built previously in chapter 3. As a recap, we mention the use of every layer that we implemented in the model as follows:

Convolution Layers

The following command creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. For the first layer, we have to specify the input shape or data format, in this case, the format is shown with the mentioned command as follows:

$$input_shape = (64, 64, 1)$$

Specifying the height and width of the input images 64x64 respectively, and the number of channels which in this case is one channel. Usually, Red-Green-Blue (RGB) images have three channels: one for each red, green, and blue colors. However, the proposed CNN classification model requires input images to be grayscale, hence we need only one channel to represent black and white. We also specify the number of filters to be learned by the model on every convolution layer. The first convolution layer will learn

64 filters from the image, the second and third convolution layers will learn 128 filters. Lastly, we will set the kernel size of the first layer to (5,5) which specifies the height and width of our 2D convolution window. Each layer has an activation function that is responsible for transforming the summed weight of inputs into the activation of the output in every layer. In our model, we used the rectified linear activation function (ReLU) which is a piecewise linear function that will give output to the given input directly if it is positive, and will output zero otherwise. It is considered the default activation function for neural networks since it makes the training process for the model easier and often achieves better performance. As seen in the example in Figure 4.2, the default ReLU has the threshold of zero. Any values above the threshold are kept at the output as-is. Otherwise, the output is zero.

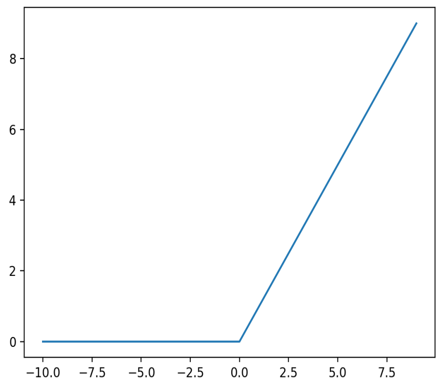


Figure 4.2: Example ReLU Activation Output Graph [26]

Max Pooling Layers

the main purpose of max-pooling layers is to down-sample the input into the window of (2,2) for each dimension along the feature axis. The window is then shifted by "strides" (which defaults to the given window size (2,2)) in each dimension thus the resulting output of each maximum pooling layer given a "valid" padding type as:

$$output\_shape = \frac{(input\_shape - pool\_size + 1)}{strides}$$

Classification

Classification technique is important to identify the malware by a standardized system. After setting up the data and the CNN model, we then test it against the dataset. After training, the model should be able to predict if the given grayscale image of a binary file is categorized as malicious or benign. We expanded our classification categories according to the list (1) mentioned previously, meaning that we have two classes for IoT malware; Mirai and Gafgyt, and one other class that represents goodware or benign binary files. The model should be able to also identify to which malware family the malicious binary file belongs, further enhancing the usefulness of the classification model.

Tensorflow Lite

The main contribution of this chapter is to convert the explained CNN deep learning model into a light version that can run on IoMT devices to achieve malware detection on the edge level. For that to be done, we will use a tool called Tensorflow Lite Converter that converts a regular Tensorflow model into a Tensorflow Lite model that is designed to run on IoT devices [27]. We will first be training the regular model over the dataset in [25] that contains binary files of around four thousand IoT specific malware. We will also be including one thousand goodware files in our training set. The training of the regular Tensorflow model will be done on a PC with the following model compile settings:

- epochs = 5
- Training dataset is 0.7% out of acquired dataset
- Testing dataset is 0.3% out of acquired dataset

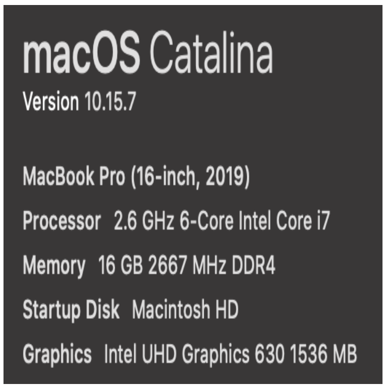
The mentioned compile settings are selected because they produce the most optimal accuracy as per our experiments in [14] more details can be found in the result section of chapter 3. Where we noticed that anything more than five epochs will not improve the accuracy but rather lower it. And the ratio of 70% training set, 30% test set provides quick training and prediction time that maintains the most optimal accuracy for the proposed model as we explained in [14]. After training, we will save the model and deploy it on capable nodes like a PC or a Server for malware detection. This model will also be the input to the Tensorflow Lite converter. The Tensorflow Lite converter will take the trained model that we produced in the previous step and convert it into a lighter model with the ability for it to run natively on IoT devices. The produced Tensorflow Lite model is expected to maintain relatively similar accuracy compared to the regular full-fledged Tensorflow model produced in chapter 3. We now showcase the accuracy metrics results of both the regular Tensorflow model and the Tensorflow Lite model to compare them against each other and state-of-the-art technologies in the results section discussed later on.

Simulation Set-up Specifications

In this section, we briefly list the hardware and software specifications that we used to build and run the proposed model for both the full version (Tensorflow, PC) and the lite version (Tensorflow Lite, IoMT).

Hardware

We used a Macbook Pro 16" Laptop with the hardware specifications shown in Figure 4.3 to train the base model (.h5) that can run on PCs and servers.


macOS Catalina
Version 10.15.7

MacBook Pro (16-inch, 2019)
Processor 2.6 GHz 6-Core Intel Core i7
Memory 16 GB 2667 MHz DDR4
Startup Disk Macintosh HD
Graphics Intel UHD Graphics 630 1536 MB

Figure 4.3: Hardware Specification Used to Train the Model

As for the Tensorflow Lite model, we will use a raspberry pi 4 model B with the following specifications:

- Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- 1GB LPDDR4-2400 SDRAM
- 2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0, BLE
- True Gigabit Ethernet
- 2 x USB 3.0 Ports, 2 x USB 2.0 Ports
- Fully backwards compatible 40-pin GPIO header
- 2 x micro HDMI ports supporting up to 4K 60fps video resolution

- 2-lane MIPI DSI/CSI ports for camera and display
- 4-pole stereo audio and composite video port
- Micro SD card slot for loading operating system and data storage
- Requires 5.1V, 3A power via USB Type C or GPIO
- PoE (Power over Ethernet) enabled (requires PoE HAT)

Software

We used the main Tensorflow frameworks along side Keras which is a wrapper library that goes on top of Tensorflow to make writing ML/DL code easier and much more efficient. We used Tensorflow Lite and Tensorflow interpreter to run .tflite models on the Raspberry Pi (IoT device). And we used OpenCV to do the conversion from the binary data to grayscale images in the pre-processing stage. All of our programming was done in Python to make it simpler and much more accessible. The following list will showcase the versions of frameworks that we used to build the proposed models:

- Tensorflow (PC) => 2.3
- Tensorflow (IoT) => 2.2
- Tensorflow Lite (IoT) => 2.3
- Keras => 2.4.3
- Python 3 => 3.8
- OpenCV => 4.3
- Sklearn => 0.23.2

Results

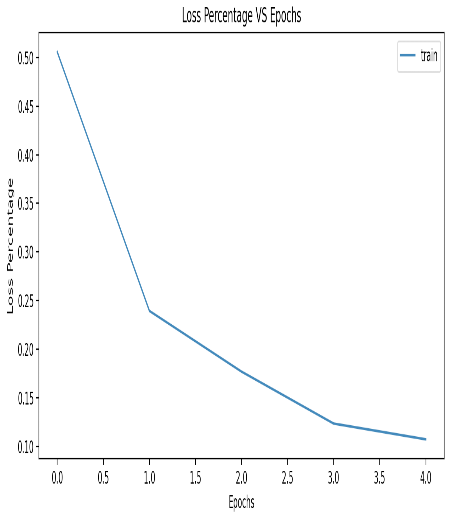
In this section, we will be showcasing the accuracy metrics produced when we evaluate the full model (PC), the lite model (IoT), and compare them against each other and similar models in the literature.

Tensorflow Model

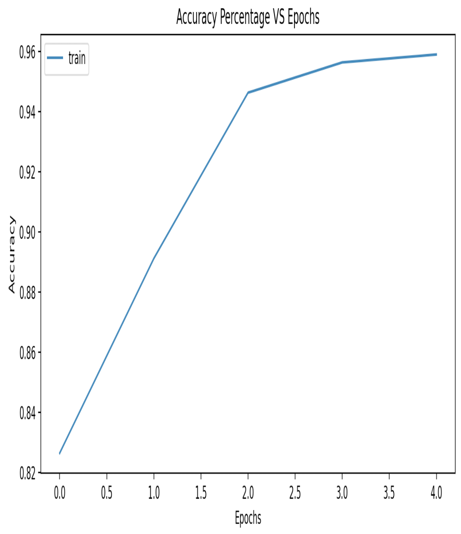
This section shows and discusses the results that we achieved in terms of model accuracy, loss function, F1-Score, recall, and precision of the model when we run the corresponding built-in Sklearn evaluation methods on the proposed model. The evaluation code is used to create the histograms of the accuracy and loss function percentages after training the model using the following code:

$$model.evaluate(test\_images, test\_labels, verbose = 2)$$

The mentioned code gave us a complete histogram graph of how well the model is trained in terms of correct classification accuracy and loss function optimization. Figure 4.4 shows the results in two graphs, Figure 4.4a explains how our model optimizes the loss function to be as minimum as possible after each epoch iteration. One epoch consists of one passing of all training dataset over the model to learn features, multiple epochs signify that we are passing the training set multiple times over the model to further enhance the classification accuracy. Figure 4.4b is the most important figure, showcasing the overall accuracy of our model after every iteration of an epoch. We can see that the model performs at an acceptable level of classification accuracy averaging around 96% on a PC. The model required an average time of $1 : 25$ minutes to finish training, and $2 - 5$ seconds to convert to Tensorflow Lite, which is considered good compared to its accuracy and the small number of convolution layers.



(a) Loss Percentage VS Epochs



(b) Accuracy Percentage VS Epochs

Figure 4.4: Tensorflow Model Accuracy Histogram on PC Node

Table 4.1: Tensorflow F1-Score, Recall, and Precision

Class	Precision	Recall	F1-Score
Gafgyt	0.99	0.98	0.98
Mirai	0.77	0.69	0.73
Goodware	0.83	1	0.91
Average	0.86	0.89	0.87
Accuracy		96 %	

Table 4.1 presents the results produced in terms of F1-score, precision, and recall values of each class when we pass the testing dataset to the evaluation algorithm that uses the model to create a list of class label predictions of the given dataset. The prediction list is then compared to the actual labels of the testing dataset to attain the mentioned values in the table. We can see that the averages are quite acceptable in terms of classification accuracy and average precision, recall, and F1-score.

Table 4.2: Tensorflow Confusion Matrix

True \ Predict	Gafgyt	Mirai	Goodware
Gafgyt	805	14	4
Mirai	7	47	14
Goodware	0	0	90

Table 4.2 shows the confusion matrix of the proposed model that highlights how many malware are identified correctly and how were miss-classified. The diagonal of this matrix shows the number of files that were correctly classified by the proposed model in their corresponding classes shown in the first column. The other columns represent the number of files that were falsely classified as the class of that row. We notice that our model had no problem classifying goodware but shows small inaccuracy when classifying Gafgyt and Mirai malware.

Tensorflow Lite Model

Similarly, we show the Tensorflow Lite version of our model in terms of classification accuracy, F1-score, precision, and recall compared to the full-fledged model. We run the metric tests in this sub-section directly on the IoT device and using only the Tensorflow Lite model to perform predictions.

Table 4.3: Tensorflow-Lite F1-Score, Recall, and Precision

Class	Precision	Recall	F1-Score
Gafgyt	0.99	0.99	0.99
Mirai	0.84	0.68	0.75
Goodware	0.83	1	0.91
Average	0.89	0.89	0.88
Accuracy		96 %	

Table 4.3 shows the metric results that we achieved by running the same testing scenario

on the Tensorflow Lite model. As we can see, the model maintained a similar classification accuracy of 96% with similar precision, recall, and F1-score values. This means that the conversion from Tensorflow to Tensorflow Lite was a seamless transition that did not harm our classification accuracy.

Table 4.4: Tensorflow-Lite Confusion Matrix

True \ Predict	Gafgyt	Mirai	Goodware
Gafgyt	801	10	2
Mirai	9	53	16
Goodware	0	0	90

In table 4.4, we present the confusion matrix of the same testing set that we used before. But this time we produced our predictions through the Tensorflow Lite model running on the IoT device directly. The achieved confusion matrix is very similar to the previous Table 4.2 which is a good sign that our model maintained its performance. However, we can see that the Tensorflow Lite model achieved slightly better classification, especially in the Mirai malware family.

Comparison

We compare our findings with the results in [28]. The latter used machine learning techniques to achieve IoT malware detection. They proposed a system that enhances IoT malware detection called SAE that boosts the classification accuracy of a given classification technique. However, if we compare the accuracy of our model to the accuracy of the base classification techniques shows promising results. The best base classification technique presented in [28] is the K-Nearest-Neighbour (KNN) technique averaging around 97.5%, followed by Support Vector Machine (SVM) at 97.5%, Decision Trees

(DT) at 96.8% which is similar to the accuracy of our model, and lastly Naive Bayes (NB) at 91.1% accuracy.

Table 4.5: Deep Learning Model Results [28]

Method	Precision	Recall	F1-Score
DT	0.968	0.967	0.968
SAE-DR	0.986	0.992	0.989
KNN	0.975	0.975	0.975
SAE-KNN	0.981	0.993	0.987
NB	0.911	0.906	0.905
SAE-NB	0.948	0.999	0.973
SVM	0.975	0.975	0.975
SAE-SVM	0.960	0.999	0.980

We can see that our model is quite similar in terms of accuracy compared to the literature. We compared our full-fledged and Lite models to the mentioned models in Table 4.5 and we can see that classification accuracy is similar. The presented literature in [28] provide models with higher accuracy and that is because they have more sophisticated CNN architecture with more layers compared to our lightweight oriented CNN that contains only three convolution layers. However, our model maintained acceptable accuracy even though we have fewer convolution layers.

Table 4.6: Deep Learning Performance for Android IoT devices [29]

Method	Accuracy	Precision	Recall	F1-Score
DexCNN	93.6%	91.8%	95.7%	0.937
DexCRNN_GRU	93.7%	91.7%	96.1%	0.939
DexCRNN_LSTM	93.4%	93.7%	93.1%	0.934
DexCRNN_BiGRU	94.4%	94.9%	93.8%	0.944
DexCRNN_BiLSTM	94.9%	93.7%	96.4%	0.950
Proposed Lite Model	96.0%	89%	89%	86%

In [29] they introduced a similar methodology to our approach in the sense that they transformed the binary of files into grayscale images to utilize CNN classification.

However, they classified a different type of IoT malware from the "Ghost" family. The presented classifier showed around 93.7% classification accuracy using Long Short-Term Memory (LSTM). We cannot fully compare our findings with [29] in terms of accuracy but our models showed higher accuracy results in classifying two IoT malware families compared to one malware family presented in [29]. However, this may be due to multiple factors ranging from the ambiguity of the grayscale images produced by the "Ghost" malware family, to the way they expand features of the grayscale image. The main advantage of the proposed model is that it requires drastically less time to train compared to the models in [29]. The DexCNN model provides an accuracy of 93.6% and requires 2 hours of training compared to $1:30$ minutes of training for the proposed model.

Chapter Conclusion

In this chapter, we proposed a deep learning CNN model that can detect IoT specific malware. We created a lighter version of this model to be deployed natively on IoMT devices achieving edge level malware detection. We produced promising results in terms of malware classification accuracy that might aid the literature to create malware prevention and security response systems dedicated to IoMT devices and applications. New cyber-security technologies to help improve the state-of-the-art of IoMT security are needed to have safer, more consistent health care systems and applications that are based on IoT [30]. Given the ever-evolving nature of security attacks, it is of utmost priority to secure IoMT applications especially those that have heterogeneous communication between IoMT devices, creating new attack surfaces for multi-vector malware attacks. With that being said, more efforts should be aimed towards creating adaptable malware detection and prevention systems that are capable of recognizing unknown and known IoT malware.

CHAPTER 5: CONCLUSION AND FUTURE WORK

Overall, machine learning and deep learning techniques proved to be an efficient utility for implementing IoT malware detection. In this thesis, we discussed multiple approaches to IoT and IoMT security in the literature. While some approaches provide robust levels of security, they are not always applicable to use in IoT environments due to processing power and energy limitations. Some approaches proposed hybrid security solutions that combine two techniques (e.g. blockchains and machine learning) in an attempt to eliminate the negatives of each technique by utilizing the positives of the other. However, the discussed techniques still require heavy computational power and can be hardly adapted to fit the needs of IoT/IoMT security. Therefore, we believe that machine learning and deep learning models that are pre-trained are a great solution to serve as IoT/IoMT malware detection tools. The adaptive nature of machine learning and deep learning provides flexibility and higher accuracy when dealing with unknown malware or zero-day attacks. However, there are still gaps in the literature that more research efforts can go towards as we mention in the following subsection.

Future Work

Our proposed models can be further optimized by acquiring more features and input data to expand the learning process. Data like attack patterns, malicious behavior of packets, and malicious network access patterns may enhance the performance and accuracy of the proposed models. The proposed model can indeed classify unknown malwares, but this is based on best effort as the classification decision relies on known malware that we used to train the models. However, certain techniques like Generative Adversarial Networks (GANs) is worth exploring to enhance the prediction accuracy of the model regarding unknown malware. It is also worth noting that we deployed our lite Tensorflow model over a Raspberry Pi 4 Model B which is considered an IoT devices but we did not deploy over real IoMT devices. Deployment on real IoMT devices might be a good opportunity for future development of this thesis.

REFERENCES

[1] S. K. Kharroub, K. Abualsaud, and M. Guizani, “Medical iot: A comprehensive survey of different encryption and security techniques,” in *2020 International Wireless Communications and Mobile Computing (IWCMC)*, IEEE, 2020, pp. 1891–1896.

[2] E. Yaacoub, K. Abualsaud, T. Khattab, M. Guizani, and A. Chehab, “Secure mhealth iot data transfer from the patient to the hospital: A three-tier approach,” *IEEE Wireless Communications*, vol. 26, no. 5, pp. 70–76, 2019.

[3] E. Yaacoub, A. Chehab, M. Al-Husseini, K. Abualsaud, T. Khattab, and M. Guizani, “Joint security and energy efficiency in iot networks through clustering and bit flipping,” in *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, IEEE, 2019, pp. 1385–1390.

[4] F. I. Salih, N. A. A. Bakar, N. H. Hassan, F. Yahya, N. Kama, and J. Shah, “Iot security risk management model for healthcare industry,” *Malaysian Journal of Computer Science*, pp. 131–144, 2019.

[5] F. Alsubaei, A. Abuhussein, V. Shandilya, and S. Shiva, “Iomt-saf: Internet of medical things security assessment framework,” *Internet of Things*, vol. 8, p. 100 123, 2019.

[6] K. Abualsaud, M. Mahmuddin, M. Saleh, and A. Mohamed, “Ensemble classifier for epileptic seizure detection for imperfect eeg data,” *The Scientific World Journal*, vol. 2015, 2015.

[7] K. Abualsaud, M. Mahmuddin, and A. Mohamed, “Wbasn signal processing and communication framework: Survey on sensing communication technologies delivery and feedback,” *Journal of Computer Science (JCS)*, vol. 8, no. 1, 2012.

[8] W. Sun, Z. Cai, Y. Li, F. Liu, S. Fang, and G. Wang, “Security and privacy in the medical internet of things: A review,” *Security and Communication Networks*, vol. 2018, 2018.

[9] Y. Zhao, “Identity-concealed authenticated encryption and key exchange,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1464–1479.

[10] L. A. Belem Pacheco, E. A. Pelinson Alchieri, and P. A. S. Mendez Barreto, “Device-based security to improve user privacy in the internet of things,” *Sensors*, vol. 18, no. 8, p. 2664, 2018.

[11] M. A. Khan and K. Salah, “Iot security: Review, blockchain solutions, and open challenges,” *Future Generation Computer Systems*, vol. 82, pp. 395–411, 2018.

[12] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, “Blockchain for iot security and privacy: The case study of a smart home,” in *2017 IEEE international conference on pervasive computing and communications workshops (PerCom workshops)*, IEEE, 2017, pp. 618–623.

[13] N. Park, J. Park, and H. Kim, “Inter-authentication and session key sharing procedure for secure m2m/iot environment,” *International Information Institute (Tokyo). Information*, vol. 18, no. 1, p. 261, 2015.

[14] A. M. N. Zaza, S. K. Kharroub, and K. Abualsaud, “Lightweight iot malware detection solution using cnn classification,” in *2020 IEEE 3rd 5G World Forum (5GWF)*, 2020, pp. 212–217.

[15] O. Franberg and A. Kung, Jun. 2019. [Online]. Available: `https://iotweek.blob.core.windows.net/`.

[16] S. S. Choi, J. W. Burm, W. Sung, J. W. Jang, and Y. J. Reo, “A blockchain-based secure iot control scheme,” in *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, IEEE, 2018, pp. 74–78.

[17] R. Kumar, X. Zhang, W. Wang, R. U. Khan, J. Kumar, and A. Sharif, “A multimodal malware detection technique for android iot devices using various features,” *IEEE Access*, vol. 7, pp. 64 411–64 430, 2019.

[18] J. Su, V. D. Vasconcellos, S. Prasad, S. Daniele, Y. Feng, and K. Sakurai, “Lightweight classification of iot malware based on image recognition,” in *2018 IEEE 42Nd annual computer software and applications conference (COMPSAC)*, IEEE, vol. 2, 2018, pp. 664–669.

[19] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, “Iotpot: Analysing the rise of iot compromises,” in *9th {USENIX} Workshop on Offensive Technologies ({WOOT} 15)*, 2015.

[20] *Virustotal-api*. [Online]. Available: `https://pypi.org/project/virustotal-api/`.

[21] Cortesi, *Cortesi/scurve*. [Online]. Available: `https://github.com/cortesi/scurve/blob/master/binvis`.

[22] S. Sharma, “Epoch vs batch size vs iterations,” *Towards Data Science*, vol. 23, 2017.

[23] [Online]. Available: `https://support.hdfgroup.org/HDF5/faq/limits.html`.

[24] M. Roopak, G. Y. Tian, and J. Chambers, “Deep learning models for cyber security in iot networks,” in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, IEEE, 2019, pp. 0452–0457.

[25] R. Vishwakarma and A. K. Jain, “A honeypot with machine learning based detection framework for defending iot based botnet ddos attacks,” in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, IEEE, 2019, pp. 1019–1024.

[26] J. Brownlee, *A gentle introduction to the rectified linear unit (relu)*, Aug. 2020. [Online]. Available: `https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/`.

[27] *Tensorflow lite converter*. [Online]. Available: `https://www.tensorflow.org/lite/convert`.

[28] F. Xiao, Z. Lin, Y. Sun, and Y. Ma, “Malware detection based on deep learning of behavior graphs,” *Mathematical Problems in Engineering*, vol. 2019, 2019.

[29] Z. Ren, H. Wu, Q. Ning, I. Hussain, and B. Chen, “End-to-end malware detection for android iot devices using deep learning,” *Ad Hoc Networks*, vol. 101, p. 102 098, 2020.

[30] J.-P. A. Yaacoub, M. Noura, H. N. Noura, O. Salman, E. Yaacoub, R. Couturier, and A. Chehab, “Securing internet of medical things systems: Limitations, issues and recommendations,” *Future Generation Computer Systems*, vol. 105, pp. 581–606, 2020.