

Transfer Learning Autoencoder Neural Networks For Anomaly Detection In Malware Infected IoT Devices



By

Unsub Shafiq

Spring-2022-MS-CS 277028 SEECS

Supervisor

Dr. Muhammad Khuram Shahzad

Department of Computing

A thesis submitted in partial fulfillment of the requirements for the degree of Masters
of Science in Computer Science (MS CS)

In

School of Electrical Engineering & Computer Science (SEECS) ,

National University of Sciences and Technology (NUST),

Islamabad, Pakistan.

(April 2022)

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis entitled "Transfer Learning Autoencoder Neural Networks for Anomaly Detection in Malware Infected IoT Devices" written by UNSUB SHAFIQ, (Registration No 00000277028), of SEecs has been vetted by the undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/M Phil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis.

Signature: _____ *M. Khuram* _____

Name of Advisor: Dr. Muhammad Khuram
Shahzad

Date: _____ 15-Apr-2022 _____

Signature (HOD): _____

Date: _____

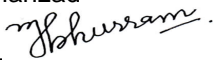
Signature (Dean/Principal): _____

Date: _____

Approval

It is certified that the contents and form of the thesis entitled "Transfer Learning Autoencoder Neural Networks for Anomaly Detection in Malware Infected IoT Devices" submitted by UNSUB SHAFIQ have been found satisfactory for the requirement of the degree

Advisor : Dr. Muhammad Khuram
Shahzad

Signature: 

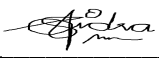
Date: 15-Apr-2022

Committee Member 1: Dr. Rabia Irfan

Signature: 

Date: 15-Apr-2022

Committee Member 2: Dr. Sidra Sultana

Signature: 

Date: 18-Apr-2022

Committee Member 3: Dr. Mehdi Hussain

Signature: 

Date: 14-Apr-2022

Dedication

To my parents, wife and sisters; their consistent support made juggling work and a Masters degree a lot easier.

Acknowledgments

Glory be to Allah (S.W.A), the Creator, the Sustainer of the Universe. Who only has the power to honour whom He please, and to abase whom He please. Verily no one can do anything without His will.

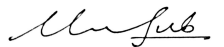
I am grateful to my supervisor, Dr. Muhammad Khuram Shahzad, for his invaluable guidance and support throughout the research activity.

Unsub Shafiq

Certificate of Originality

I hereby declare that this submission titled "Transfer Learning Autoencoder Neural Networks for Anomaly Detection in Malware Infected IoT Devices" is my own work. To the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEecs or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEecs or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics, which has been acknowledged. I also verified the originality of contents through plagiarism software.

Student Name: UNSUB SHAFIQ

Student Signature: 

Contents

1	Introduction and Motivation	1
1.1	Introduction	2
1.2	Motivation	3
1.3	Contribution	4
2	Background and Literature Review	6
2.1	Background	7
2.1.1	Distributed Denial of Services - DDoS	7
2.1.2	Bots and Botnets	11
2.1.3	Internet of Things	12
2.1.4	Autoencoders	13
2.2	Related Work	14
3	Methodology	18
3.1	Understanding the Datasets	19
3.1.1	NBaIoT Dataset	20
3.1.2	CIC-IDS2017 Dataset	22
3.1.3	Programming Language Libraries	25
3.2	NBaIoT – Anomaly Detection Transfer Learning	26
3.3	CIC-IDS2017 – Anomaly Detection	32
4	Results and Discussion	35

CONTENTS

4.1	NBaaS Dataset - Anomaly Detection and Transfer Learning	38
4.1.1	Evaluating Model Accuracy	38
4.1.2	Evaluating Model Training Time	43
4.1.3	Correlating Static Features with Model Performance	45
4.2	CIC-IDS2017 Dataset - Anomaly Detection	46
5	Conclusion and Future Work	49
5.1	Future Work	50

List of Abbreviations

Table 1: List of Abbreviations

Abbreviation	Description
ACK	Acknowledge packet in a TCP handshake
C&C / CC / C2	Command and Control
CAE	Convolutional Autoencoder
CERL	Computer-based Education Research Laboratory
CIC-DDoS2019	DDoS Evaluation Dataset
CIC-IDS2017	Intrusion Detection Evaluation Dataset
DDoS	Distributed Denial of Service
DGA	Domain Generation Algorithm
DNS	Domain Name Service
DoS	Denial of Service
HTTP	Hyper Text Transfer Protocol
IoT	Internet of Things
IPFIX	IP Flow Information Export
ISP	Internet Service Provider
LDAP	Lightweight Directory Access Protocol
LSTM	Long Short Term Memory
ML	Machine Learning
MSSQL	Microsoft Structured Query Language
NBaIoT	Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders (dataset)
NETBIOS	Network Basic Input/Output System
NTP	Network Time Protocol
PCAP	Packet Capture
PLATO	Programmed Logic for Automatic Teaching Operations
SNMP	Simple Network Management Protocol
SSDP	Simple Service Discovery Protocol
SYN	Synchronize
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
UDP	User Datagram Protocol
VAE	Variational Autoencoder

List of Figures

2.1	PLATO system	7
2.2	DDoS Classification	8
2.3	Representation of an example botnet topology	12
2.4	AutoEncoder Neural Network Layout	13
3.1	DDoS Classification	22
3.2	DDoS Classification	23
3.3	Autoencoder Pre-Transfer Learning	30
3.4	Loss Function plot of the IoT device SimpleHome-XCS7-1003-WHT-Security-Camera for Bashlite and Mira datasets respectively	30
3.5	Autoencoder Post-Transfer Learning	32
4.1	MIRAI Transfer Learning Results	39
4.2	BASHLITE Transfer Learning Results	39
4.3	MIRAI to BASHLITE Transfer Learning Results	40
4.4	BASHLITE to MIRAI Transfer Learning Results	40
4.5	Anomaly Detector’s performance change for MIRAI dataset	43
4.6	Model Training times for Mirai dataset	45
4.7	Model Training times for Bashlite dataset	45
4.8	Percentage Feature Overlap of IoT devices	46
4.9	A representation of how some malicious traffic does not have significant variance in network footprint	47

LIST OF FIGURES

4.10 CIC-IDS2017 model performance results	48
4.11 CIC-IDS2017 with different optimization algorithms	48

List of Tables

1	List of Abbreviations	viii
3.1	Dataset attributes	20
3.3	NBaaS dataset feature description	21
3.4	A possible discrepancy in the Aggregation time units	22
3.5	CIC-IDS2017 Dataset size	24
3.6	Representation of the Dataset size and its consumption during training stages	31
4.1	Experimental scope against each accuracy matrices	38
4.2	Percentage decrease in model accuracy before and after transfer-learning (TL) across device	42
4.3	Percentage decrease in model accuracy before and after transfer-learning (TL) across device and malware type	42
4.4	Time saved when transfer-learning on Mirai dataset	44
4.5	Time saved when transfer-learning on Bashlite dataset	44
4.6	IoT Device Feature Distribution in NBaaS dataset	45

Abstract

Distributed Denial of Service (DDoS) attacks have persisted against defensive measures with their sheer capability of obscurity and the simplicity of the attack vectors that they exploit, i.e. exhausting the victim’s computing resources. The advent of Internet-of-Things (IoT) has led to a massive increase in smart internet-connected devices that often have customized firmware with limited and irregular security patches. This has made them targets for hosting bot malware and can contribute to traffic in a DDoS attack. Many researchers have worked on developing anomaly detectors to identify possible infected hosts and have incorporated ML models within their techniques as well. However, the ubiquity of IoT devices has made training ML models on a per-device and per-malware basis impractical.

In this thesis, we focus on the autoencoder neural-network-based anomaly detection technique and evaluate the efficacy of the transfer-learning technique in reducing the training time of anomaly models for IoT devices. We base our hypothesis on the intuition that similar IoT devices should have a similar network footprint and therefore, the latent representation of network footprint should be transferable across devices. The study bases itself on the NBaIoT [1] dataset, which consists of 115 traffic features of real IoT devices infected with Mirai and Bashlite. We observe that while the accuracy of an anomaly model decreased when tested against data from a new IoT device, re-training the innermost layers of the autoencoder with at least 10% of the available dataset restored the anomaly model’s performance. We further evaluate the capability of autoencoders against the CIC-IDS2017 dataset; consisting of network-flow information derived from PCAP data, which can be considered more synonymous with IPFIX/Net-flow record formats prevalent in the industry.

CHAPTER 1

Introduction and Motivation

Chapter Summary

The internet, since its inception, has led to explosive growth in digital space. However, with the world increasingly adopting the digital revolution, cyber-attacks are becoming more frequent. We introduce how botnets and Internet-of-things have contributed to the rise of DDoS cyberattacks and the challenges of using machine learning in mitigating them.

1.1 Introduction

The history of digitization has seen exponential growth in services and products that are increasingly dependent on computing systems. Composed of increasing computational capability and faster interconnect mechanisms, we are now in an era where every information and service across the globe is available instantaneously and is now known as Cyberspace. Examples to give perspective include monitoring one's home with smart cameras while vacationing in another country, offering consultancy services to customers halfway across the world, and getting things delivered to your doorstep without ever setting foot in a physical store.

We shall limit the scope of discussion to the public internet within this thesis. Understanding these devices is fundamental since a large part of Cybersecurity deals with protecting these devices from running malicious software and/or generating unwanted traffic.

With the adoption of internet-based service offerings, there was a consequent rise in mechanisms that try to knock any such service offline [2]. A crude but very effective way to do so is to overwhelm such service with unwanted traffic, so much so that legitimate traffic does not get served; now known as the Denial-of-Service (DoS) attack. However, this faced two major problems;

- Generating huge traffic from a single source (device) is not feasible and;
- A single source can be easily identified and blocked without many business implications.

The concept of bots was not conceived entirely to practice malicious activities; 'good bots' have been and are currently in use in order to make the internet a better experience. Search engines have widely used bots to traverse the web and index the contents. However, the achievements of these deployments vastly go unnoticed whereas the disruptive actions of malign botnets gain traction due to their sheer scale and sudden impact on digital commerce.

The term 'bots' refers to computing devices that have been compromised with malicious code and can be instructed to perform a particular action remotely. Bots eliminate the need for an attacker to set up systems dedicated to attack generation; masks the true

attacker and distribute the sources significantly such that it becomes difficult to differentiate between a legitimate source and a compromised source. DoS attacks manifesting from such diverse sources are thus known as Distributed-Denial-of-Service (DDoS) attacks.

Advancements in anti-malware software have made infecting a computing device increasingly difficult and have impacted the rate of propagation of such malware. But even these shields have a chink in their armor. The vast majority of them are capable of running on only full-fledged operating systems and use known signatures to match malicious code files. This covers only a limited subset of the computing devices, mainly personal computers, mobile phones/tablets, and to some degree enterprise-grade servers. One large category that goes unprotected is the Internet-of-Things (IoT). These devices have proprietary firmware and are designed to execute limited functions while communicating over the internet.

IoT devices do not have the capacity [3] of running resource-hungry anti-malware software and thus cannot be protected via local defense mechanisms. At the same time, new applications such as high-resolution video surveillance require faster computing chips and higher-bandwidth connectivity in IoT devices. These two capabilities combine to make them an ideal target to serve as bot hosts. Coupled with irregular firmware upgrades and easy-to-decipher default-access credentials, such devices are often prone to vulnerabilities that can be exploited and used for the propagation of bot malware. IoT-based botnets started appearing more than a decade ago with the likes of Mirai [4] and Bashlite achieving international fame due to their massive scale. Researchers have since worked on alternative ways for detecting bot activity in a network, one dimension of it being anomaly detection. With Machine-Learning (ML) being a hot industry topic in recent years, many have put efforts into evaluating the efficacy of machine learning methods in detecting bot behavior and leveraging it for anomaly detection.

1.2 Motivation

DDoS attacks generated by botnets are a persistent threat that is still prevalent today. The obscurity of Command-and-Control (CC) mechanisms and their distributed nature are among the major reasons why malign botnets have thrived till now. DDoS attacks originated by botnets not only cause disruption and losses to their victims but also im-

pact the hosting ISPs and smaller networks with which the actual compromised devices reside. Losses caused by having bots in one’s network can be multi-dimensional, all of which imply business impact. These include;

- A breached perimeter; whereby an attacker already has control over a device on the private network. And thus can move laterally [5], generating insider attacks and infecting more hosts.
- Information theft, such as credentials, confidential and private data [6].
- Loss of computation cycles, impacting legitimate services running on the infected host.
- Degradation in network reputation [2], example IP blacklisting to due malicious activity originating from it.

Realizing the need for organizations to proactively protect their network infrastructure from originating malicious traffic, researchers have proposed a variety of ways to detect anomalous behavior. Diverse proposals have been made that attempt to detect anomalous behavior at a variety of stages; propagation of bot binaries, communication attempts with CC, and anomalous traffic generation. However, training complex models as part of priming the anomaly detector suffer from a lack of available data on benign behavior and high cost and effort in model training and optimization. Coupled with the increasing variety of IoT devices, training anomaly detectors on a per device basis is impractical.

1.3 Contribution

In accordance with the historical background and problem statements mentioned above, we summarize the research objectives and respective contributions of this thesis.

Q-1: While the similarity in traffic patterns of similar IoT devices (manufactured by different vendors) is implied, can this be reflected in the encoder part of the autoencoder neural network?

Contribution: We trained an autoencoder-based anomaly detection model on a per-device and per-malware basis and then tested it against the benign and malicious data

of other IoT devices. Observations on a relative decrease in model accuracy were noted. Transfer-learning the same anomaly model showed a marked improvement in prediction accuracy, an average of 9.52% for Mirai and 44.59% for Bashlite. Additionally, we demonstrate that Transfer-learning still holds true for cases in which a model is transferred across malware of similar nature.

Q-2: Can the static hardware features of IoT devices be co-related to their expected traffic patterns?

Contribution: Our observations in this regard are partially conclusive. In a majority of cases, we observed that a model trained on a more feature-rich IoT device generally performed well against un-seen *normal* data of an unknown device. We conclude that enumerating the hardware features of the IoT devices can only be a partial contributor to predicting the expected benign traffic. Soft features of the firmware/OS may have a more significant role in this regard.

Q-3: Exploring the capability of autoencoders as anomaly detectors, against a variety of network data formats.

Contribution: We leverage the CIC-IDS2017 dataset containing network footprints of a variety of intrusion attempts. While autoencoders show promise in identifying traffic patterns that vastly differ from the benign behavior (due to a large offset in reconstruction error); other malicious traffic with a “normal-like” network footprint can pass through without being flagged. This might be improved by introducing new features such as IP reputation, whether destination ports are well known or not etc.

CHAPTER 2

Background and Literature Review

Chapter Summary

This section introduces the growth of Botnets and IoT over the past decades and how they can combine to deliver some of the most lethal DDoS attacks experienced in recent years. This challenge has garnered significant research interest, ranging from detecting botnets on a global scale to placing convenient anomaly detectors in enterprise environments in order to detect and block hosts infected with malware. The domain of machine learning is being actively explored to develop new anomaly detection engines. We introduce how the autoencoder neural network has been used so far in the battle against botnets.

2.1 Background

2.1.1 Distributed Denial of Services - DDoS

The objective simplicity of a DoS or DDoS attack is intriguing; it aims to simply knock a resource “*offline*”. This does not need to be taken in a literal sense. Any resource not able to respond to legitimate user requests is deemed “*offline*”, and this effect can be achieved simply by engaging a resource with non-productive tasks exhausting its capacity.



Figure 2.1: PLATO system

The earliest well-known instance of a DoS attack is attributed to David Dennis in 1974 when he “*accidentally*” found out that running the external command on a PLATO terminal that had no external devices connected froze the terminal and required a hard reset to recover. Coupled with the default configuration where this command could be sent to a remote terminal, David was able to write a program that executed this command on all terminals at the lab in CERL (Computer-based Education Research Laboratory), University of Illinois Urbana-Champaign.

This piece of history has quite of few lessons in it for us;

1. It shows the importance of programming logic and how it must be developed keeping in mind possible abnormal behaviors; in this case, running the external command before plugging in the external device which is clearly in the wrong order.
2. It shows how capabilities thought of as “features” can be exploited with mal-intent; in this case, the ability to remotely execute a command on a terminal.
3. it is indicative of the curious and opportunistic nature of attackers. After all, David Dennis was only 13 years old at the time.

While David Dennis exploited the logic of PLATO system, the early days saw a gener-

ation of bandwidth-based DoS attacks by computing enthusiasts simply to one-up each other on a variety of platforms. It was not until the late 1990s did the first large-scale DDoS attacks start surfacing. These network-based DDoS attacks have significantly grown over time and the notoriety of such attacks has only increased ever since; and are still a persistent threat today.

Network-based DDoS attacks target the capacity of the network stack of the endpoint or any in-line network element. Such attacks attempt to exhaust the buffer or session store capacity of the endpoints or of the network elements through which a particular endpoint is reachable. Such attacks can be classified as ‘Reflection Attacks’ or ‘Exploitation Attacks’ [7].

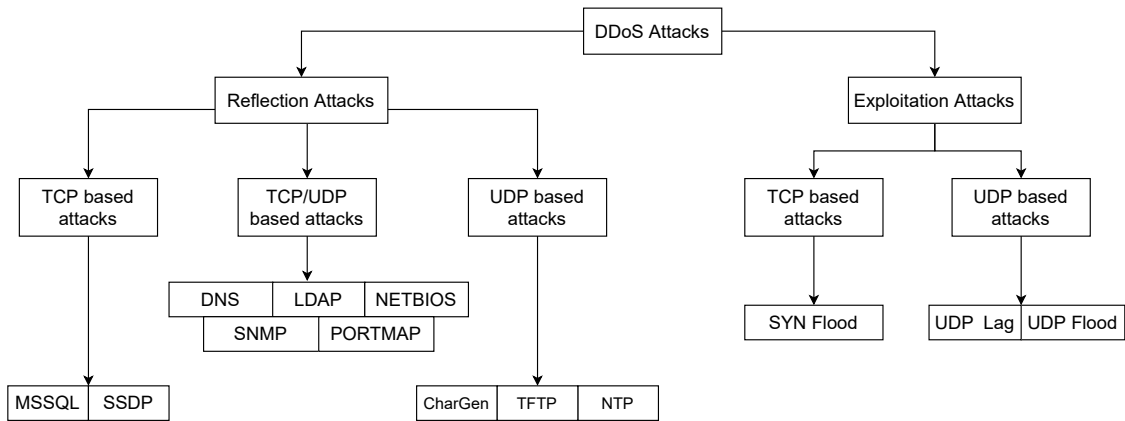


Figure 2.2: DDoS Classification

Reflection attacks primarily flood the victim with unwanted responses to the queries the victim never asked. For example, an attacker may query a DNS server for a particular record, spoofing the source IP such that when the DNS server responds, it sends the response to the victim. Now the victim needs to inspect the received packet, identify that it is not relevant, and then discard it. Flooding the victim with such traffic shall eventually occupy the victim in discarding unwanted packets rather than serving actual legitimate requests. Often the response sizes are many-fold larger than the query itself; hence reflection attacks can not only hide the original attacker but also achieve an amplification in bandwidth consumed when generating the attack.

Exploitation attacks instead attempt to misuse the algorithmic logic with the intent of overwhelming the server. For example, the SYN flood attack attempts to exploit the fact that an endpoint stores the state of a half-open TCP connection while waiting for

the three-way handshake of TCP to complete (i.e SYN, SYN-ACK, ACK). The attacker in this case initiates a flood of SYN packets forcing the endpoint to exhaust its capacity of maintaining the state of half-open sessions. A brief description of the attacks listed in Figure 2.2

REFLECTION ATTACKS

- **MSSQL(TCP)**: This attack manifests by exploiting the Microsoft SQL Server Resolution Protocol's responses to the client query. Attackers subject the MSSQL server with queries containing spoofed IP addresses.
- **SSDP(TCP)**: The Simple Service Discovery Protocol is widely used among Universal Plug-and-Play devices to advertise their existence in the network. Other endpoints can then query the UPnP devices the list details of the services they can offer. This attribute is exploited by attackers who generate a flood of such requests with spoofed source addresses; such that the responses are sent to the target victim.
- **DNS(TCP/UDP)**: The Domain Name Service responds to queries of translating domain names to IP addresses. Responses to DNS queries can vary in size with some records containing relatively large content (such as public keys in the DKIM records). Attackers query for such records with spoofed addresses such that responses are sent to the victim while achieving amplification as well.
- **LDAP(TCP/UDP)**: LDAP (Lightweight Directory Access) is a widely used protocol for directory access. It is widely used in authentication and maintaining a registry of users, endpoints, services, etc. Attackers achieve a high amplification rate when exploiting the LDAP protocol in order to flood the target victims.
- **NETBIOS(TCP/UDP)**: NETBIOS name service is a common presence in environments running Microsoft's Windows Operating services. Attackers query the service with spoofed addresses, thereby flooding the victim with unwanted and amplified response traffic.
- **SNMP(TCP/UDP)**: The Simple Network Management Protocol is a common protocol used to collect statistical / performance information from entities such as switches and routers. SNMPv2 used a "community_name" preshared as an

authentication mechanism that was susceptible to brute-forcing. Responses to SNMP queries are significantly larger and cause high amplification.

- **PORTMAP(TCP/UDP)**: Portmapper is a mechanism to which Remote Procedure Call (RPC) services register in order to allow for calls to be made to the Internet. When a client is looking to find the appropriate service, the Portmapper is queried to assist. This means, that when it is queried, the response size varies wildly depending on which RPC services are operating on the host. Attackers can exploit these large response sizes to generate an amplified DDoS attack on the victim.
- **CharGen(UDP)**: The Character Generator Protocol runs on UDP port 19 and is often found to be used by internet-enabled printers and copiers. This protocol is often used for troubleshooting purposes.
- **TFTP(UDP)**: A Trivial File Transport Protocol exposed to the internet on UDP port 69 can be used as a reflector and amplifier for generating attacks on victims.
- **NTP(UDP)**: In an NTP (Network Time Protocol) amplification, an attacker uses a spoofed IP address of the victim's NTP infrastructure and sends small NTP requests to servers on the Internet, resulting in a very high volume of NTP responses.

EXPLOITATION ATTACKS

- **SYN Flood(TCP)**: A SYN Flood, often generated by botnets, is designed to consume resources of the victim server, such as firewalls or other perimeter defense elements, in an attempt to overwhelm their capacity limits. The target receives SYN packets at very high rates which rapidly fill up its connection state table, resulting in disconnections and dropping of legitimate traffic packets.
- **UDP Lag(UDP)**: UDP Lag is a particularly popular technique among multi-player gamers where an attacker attempts to induce an artificial lag on the victim by periodically flooding and consuming its bandwidth. This attack is timed such that the lag is large enough to cause a disruption in the game but not large enough to disconnect the victim from the game servers.

- **UDP Flood(UDP):** In a UDP Flood, attackers send small spoofed UDP packets at a high rate to random ports on the victim’s system using a large range of source IPs. This consumes essential network element resources on the victim’s network which is overwhelmed by a large number of incoming UDP packets.
- **HTTP/S Flood(TCP):** HTTP and HTTPS is a transport protocol for browser-based Internet requests, commonly used to load webpages or to send form content over the Internet. In an HTTP/S flood attack, the attacker exploits seemingly-legitimate HTTP GET or POST requests to attack a web service or application. These attacks often utilize many botnets such as infected IoT devices.
- **HTTP Slowloris:** Unlike the HTTP flood attack, the HTTP slow-loris attack attempts to hog as many connections to a Web Server as possible. This is done by utilizing partial HTTP requests and then attempting to keep the HTTP session active for as long as possible. This method attempts to exhaust the endpoints’ capacity of sessions in a much more subtle way.

2.1.2 Bots and Botnets

The term “Bot” refers to any device containing malware that allows it to be “*instructed*” from a remote Command-and-Control (C&C) entity. While in the early days, such malware was targeted towards Desktop systems; with the proliferation of smart internet-connected devices, the threat landscape has increased significantly [8]. This is evident in the many bot malware prevalent today that target Internet-of-Things (IoT) devices. The pool of Bots under the control of a single CC is referred to as a “botnet”; a term synonymous with the term “internet”, signifying the medium of connectivity that the bots and CC have. The protocols of the internet form the various mechanisms for bots to discover and connected to their CC, and for CC to send instructions to bots. Internet bots can be traced back to the late 1980s [3], during which the Internet-Relay-Chat (IRC) formed the medium of choice for connectivity. While some of the initial bots were seemingly innocuous, such as keeping an IRC chat active so that the server did not close it due to inactivity; variants quickly emerged that were Trojans or Worms that were installed onto the client machines and used IRC channels to listen for instructions.

However, not all bots are designed with mal-intent. “Web-Crawler”, created in 1994,

was the first bot created to index the web. “GoogleBot” was created in 1996 and has since then evolved into the giant we know today.

After the early 2000s, which saw some of the first notable botnets, the prevalence of botnets has grown. The Storm [5] botnet is estimated to have infected almost 50 million devices during its lifetime and is suspected to have been used for a variety of crimes such as identity theft and stock price fraud. Bots have also been used to generate spam emails, the most notable among these was the Cutwail botnet that notoriously sent out over 70 billion emails per day. This shows the extent of the negative impact botnets can have on digital commerce. However, among these, DDoS generating botnets have the most disastrous impact, although relatively short-lived but capable of causing blackouts on the internet.

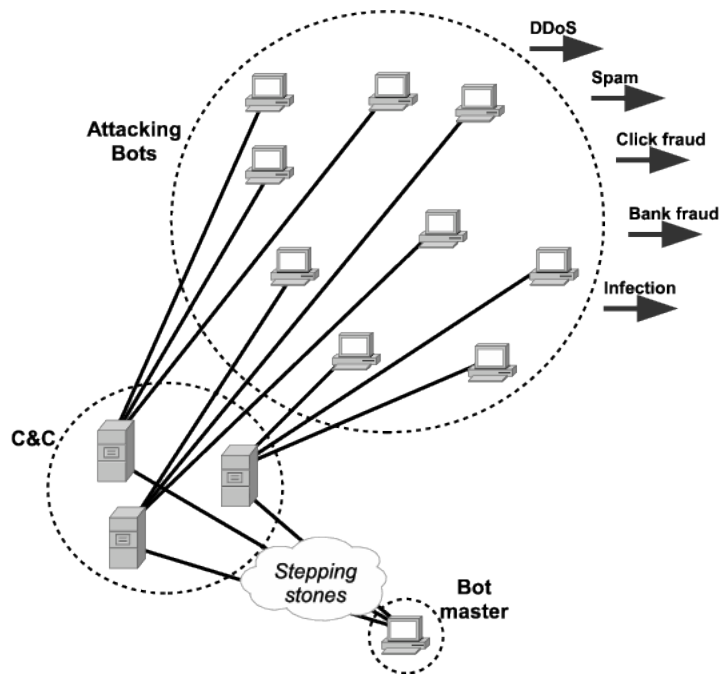


Figure 2.3: Representation of an example botnet topology [6]

2.1.3 Internet of Things

A multitude of factors such as the rapidly decreasing cost of computing hardware and better software development practices have fueled the exponential digitization of our world. Newer use-cases require real-time access and control of devices such as sensors and actuators. As a result, newer components are now capable of connecting to the

internet as well and have led to the growth of the term Internet-of-Things. Devices such as baby monitors, doorbells, and surveillance cameras connect to the internet in order to give remote management capabilities as well as serve live feeds.

While the rise of IoT has resulted in an increase in innovative new solutions and services, it has also developed into a vast attack surface that has become easy prey to botnets. The proliferation of IoT has had little regard for mechanisms that deliver adequate security patches. Coupled with poor firmware development practices and easy to identify default credentials, IoT devices have been susceptible to compromise and integrated into large-scale botnets.

2.1.4 Autoencoders

Autoencoders are neural networks that force the network to learn about the most important features of observation, such that the original observation can be re-constructed from them with minimal error. Autoencoders have been leveraged as compression engines for images and other high-dimensional datasets. Their unique bottleneck shape makes them very good at learning about a specific set of patterns, a deviation from which causes a large reconstruction error. This quality has been explored as an indication of anomaly by many researchers across a variety of domains.

Figure 2.4 contains a representation of a typical autoencoder neural network.

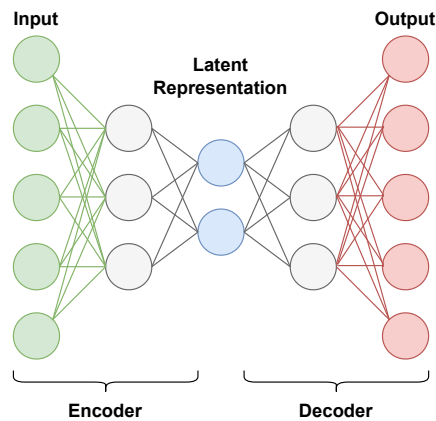


Figure 2.4: AutoEncoder Neural Network Layout

The loss function of the autoencoder neural network is a representation of the difference between the original data received on input and the reconstructed data from the latent

representation. The mean-squared error of the two data records is widely used.

$$L(x, x_R) = 1/n \sum (x - x_R)^2 \quad (2.1.1)$$

Here x refers to the original data vector, and x_R refers to the vector reconstructed from the latent representation.

2.2 Related Work

In the previous section, we introduced IoT, botnets, and DDoS since they often go hand-in-hand in large operations that use DDoS in their attacking arsenal. IoT devices have become prime targets for bot infection and the ubiquity of IoT devices makes them ideal sources to generate DoS traffic.

Extensive work on anomaly detection techniques has been undertaken by researchers over the past two decades. With the popularization of Machine Learning techniques and their capability of learning how to differentiate between non-linearly distributed data, researchers have extensively explored various ML models as potential anomaly detection engines. In particular, efforts have been made on learning the normal and anomalous behavior in the network as a means of indicating whether a particular endpoint is compromised.

The fundamental nature of botnets, i.e consisting of widely dispersed peers and command-and-controls across the internet with masked communication methods [9][10], means that there is no single sure-shot way that may be taken to cease all bot activity without hampering legitimate traffic.

Khattak et al. [11] have documented an excellent taxonomy of the lifecycle of a botnet. A botnet's life cycle starts with the propagation of bot binaries. The propagation phase's end objective is to have the bot malware installed into as many systems as possible. And a variety of mechanisms can be used to this end that may or may not require human intervention. Bot malware such as Mirai actively scans for vulnerable devices on the network, looking for devices allowing unauthenticated access or using insecure/default credentials [12]. Once breached, a small bootstrap code is run that then downloads the complete binary from the Command-and-Control (CC). Other propagation methods include the wide use of phishing emails and offerings of freeware in order to dupe users

into installing the malicious bot binary into their systems.

Equally important to the distribution mechanism of the bot binaries is to ensure it bypasses antivirus software which usually uses signature-based detection methods. *Storm* botnet [13] was found to be re-encoding its malware twice every hour for this purpose. However, botnets targeting IoT devices could conveniently overlook this complexity since such devices do not have the computing power necessary to run complex anti-virus software. The next phase after infection constitutes establishing a covert mechanism of receiving instructions from the CC, often referred to as the rallying phase [14]. The prime objective in this phase is to hide the identity of the CC and to ensure that instructions passed down to the bots are encrypted. Mechanisms include using a "fast-flux" method where the CC server's addresses are quickly rotated behind a DNS name (Storm); leveraging domain-generation-algorithms (DGA) [15][12] where each newly infected machine attempts resolution of randomly generated domain names in order to discover its CC. Newer variations have exploited peer-to-peer mode of communication that further obfuscates the CC [11][12].

A large number of infected machines can be used for a number of malpractices [16] that include spying, stealing personal information, and using available compute resources to attack other resources/services on the internet in the form of spam, DDoS, etc. The latter in particular has been used to generate large-sized DDoS attacks and constitutes a persona easily identifiable in the network. Constant evolution in the techniques of establishing botnets has kept researchers in a race to identify new mechanisms of identifying bot activity. Research in this regard has turned to leveraging Machine Learning techniques to detect bot activity at different stages of bot infection, i.e propagation, rallying, and post-infection behavior [17].

Kate et al. [18] targeted the identification of bot malware that used Domain-Generation-Algorithms (DGA) based domain names for finding its respective CC. Such malware creates anomalous DNS traffic during the rallying phase. They leveraged the deterministic nature of such algorithms and trained a deep neural network composed of LSTM, CNN, and ANN in order to identify whether a particular host was making DNS calls for domains that were DGA generated. In a similar study, Tu et al.[19] leveraged the similarity of DNS queries in order to identify bot-infected machines.

Rohan et al. [20] evaluated the detection of DDoS traffic from consumer IoT devices

by various supervised learning models. They found that K-Nearest neighbors, random forest, and neural-net models were the most effective classifiers of anomalous traffic. In this case, the model is designed to detect when actual DDoS traffic gets generated.

Autoencoders [21] are a neural network that is trained to reconstruct their input. Their main purpose is learning in an unsupervised manner an “informative” representation of the data that can be used for various implications such as clustering. An important tradeoff in autoencoders is the bias-variance tradeoff. On the one hand, we want the architecture of the autoencoder to be able to reconstruct the input well (i.e. reduce the reconstruction error). On the other hand, we want the low representation to generalize to a meaningful one. Throughout the years, autoencoders have been used for a variety of use cases, including clustering, dimensionality reduction, and anomaly detection.

In order to deal with the inherent trade-offs required by autoencoders, multiple approaches have been demonstrated. Researchers have introduced sparsity in the hidden activation layers of the neural network, hence coining the term "Sparse Autoencoders" that ensure that the learned representation is not an identity function [22][23]. This can be done in conjunction with the *bottleneck* approach or replace it entirely. A major improvement in the representation capabilities of autoencoders has been achieved by the Variational Autoencoder Model (VAE). Following the Variational Bayes Inference, VAE is a generative model that attempts to describe data generation through a probabilistic distribution. An et al. [24] evaluated the performance of a VAE based anomaly detection approach on the KDD and MNIST dataset.

While Autoencoder [25] neural networks have already established themselves as quite capable in areas such as image reconstruction and denoising, researchers are now extensively leveraging its capabilities as an anomaly detector in IoT.

Hwang et al. [26][27] in their two papers have demonstrated the use of an autoencoder for early network anomaly detection and to perform unsupervised malicious IoT traffic classification, an unsupervised clustering use-case. Their experimentation revealed a performance of nearly 100% and a very low false-positive rate of 0.83%. Salahuddin et al. [28] have demonstrated the use of autoencoders on the CIC-IDS2019 [29] and achieved an F1-Score of 99% for a majority of devices and greater than 95% for all devices.

Chen et al. [30] demonstrate the application of Convolutional Autoencoder (CAE) to

perform the dimensionality reduction. As the Convolutional Autoencoder has a smaller number of parameters, it requires less training time compared to the conventional Autoencoder. By evaluating on NSL-KDD dataset, the CAE-based network anomaly detection method outperforms other detection methods.

CHAPTER 3

Methodology

Chapter Summary

This section presents a brief on the research methodology; this includes an introduction to the datasets, the workflow of the machine-learning and transfer-learning implementations, and a discussion on the rationale for each step.

Existing papers [31][11] have used variations of the autoencoder models for the classification of traffic types using various datasets including the CIC-IDS2017. Many of the current researchers opt for advanced forms of autoencoders such as variational autoencoders (VAR) and sparse autoencoders. While these researchers have demonstrated good accuracies in detecting intrusion attempts, almost all of them train their neural networks on known and labeled malicious data. In the scenario for this thesis, we assume that only benign network behavior is available for a device. Further, while discussions on transfer learning are flooded with examples on CNN-based models where only the last classifier layer is changed, we wanted to explore whether this could be done to ANN models. In this regard, the simplicity of a simple autoencoder neural network helped build an intuitive sense of the hypothesis.

3.1 Understanding the Datasets

In our selection of datasets, we looked for the availability of the following qualities;

- Fairly recent datasets, representative of modern-day DDoS attack vectors and bot infection.
- Labelled records representing “benign” traffic on a per device basis. This was essential since our autoencoder anomaly models were to be trained to perform extremely well on normal traffic data.
- A dataset where the device was the originator of anomalous traffic.
- A dataset structure that was as close as possible to real-life data.

We chose to use the NBaIoT [1] and CIC-IDS2017 [7][32] datasets for our research. Both of these datasets were created fairly recently and consisted of labeled “benign” records on a per device basis. However the datasets also have a few fundamental differences; while the NBaIoT dataset consists of data on DDoS attacks being generated by a variety of IoT devices, the CIC-IDS2017 consists of victim endpoints towards which DDoS attacks are targeted. On the other hand, the CIC-IDS2017 dataset records are network flows which is a much closer representation of the IPFIX [33] standard used in the industry. The following table illustrates how each dataset fulfills the above criteria.

Index	DataSet Name	Fairly Recent Dataset	Labelled “benign” records	Originator of DDoS	Closeness to industry standard IPFIX
1	NBaIoT	✓	✓	✓	✗
2	CIC-IDS2017	✓	✓	✗	✓

Table 3.1: Dataset attributes

Another recent dataset specifically tailored to DDoS is the CIC-DdoS2019 [29], however, this dataset does not contain labeled instances of normal data traffic. Due to this reason, this dataset did not set well with the objectives of our thesis which aim at learning the benign behavior of the device and its transfer-ability to other devices of varying similarity.

3.1.1 NBaIoT Dataset

The NBaIoT dataset [1] was generated in a lab environment consisting of a multitude of IoT devices. Network traffic data was captured before and after infecting each device with two very well-known malware, Mirai and Bashlite. This data was passed through a feature extraction process that created 115 numeric features.

The dataset consists of nine IoT devices with data segregated among three categories, benign, malicious-Bashlite, and malicious-Mirai. The following table summarises the dataset size for each device.

The datasets were extracted from a raw packet capture and aggregated first by source-IP/MAC/port and then by time. Splitting the raw data into smaller chunks allows for the extraction of statistical attributes such as mean, variance, etc for that duration. This also allows for the detection of sudden short-lived spikes in malicious traffic and whether it sustains for a longer period of time or not. The following table summarizes the first phase of aggregation and its co-relation with the feature extraction.

Aggregated By	Denoted By	Statistic	Example Column Name	Unit
Source-IP	H	mean	H_L5_mean	size (bytes)
		variance	H_L5_variance	size (bytes)
		number	H_L5_weight	count (integer)
Source-MAC-IP	MI	mean	MI_dir_L5_mean	size (bytes)
		variance	MI_dir_L5_variance	size (bytes)
		number	MI_dir_L5_weight	count (integer)
Channel	HH	mean	HH_L5_mean	size (bytes)
		variance	HH_L5_std	size (bytes)
		number	HH_L5_weight	count (integer)
		magnitude	HH_L5_magnitude	size (bytes)
		radius	HH_L5_radius	size (bytes)
		covariance	HH_L5_covariance	size (bytes-squared)
Channel-Jitter	HH_jit	Corelation-coefficient	HH_L5_pcc	Number (b/w +1 and -1)
		mean	HH_jit_L5_mean	time (ms)
		variance	HH_jit_L5_variance	time (ms)
Socket	HpHp	number	HH_jit_L5_weight	count (integer)
		mean	HpHp_L5_mean	size (bytes)
		variance	HpHp_L5_std	size (bytes)
		number	HpHp_L5_weight	count (integer)
		magnitude	HpHp_L5_magnitude	size (bytes)
		radius	HpHp_L5_radius	size (bytes)
Corelation-coefficient	HpHp_L5_pcc	covariance	HpHp_L5_covariance	size (bytes-squared)
		Corelation-coefficient	HpHp_L5_pcc	Number (b/w +1 and -1)

Table 3.3: NBaIoT dataset feature description

Assuming that time frames are ones as mentioned in the paper.

<i>Time-frame in data header column</i>	<i>As in paper</i>	<i>Perceived from naming</i>	
L0.01	100ms	10ms	10ms
L0.1	500ms	100ms	100ms
L1	1.5s	1s	1min
L3	10s	3s	3min
L5	1min	5s	5min

Table 3.4: A possible discrepancy in the Aggregation time units

For each feature listed, statistical values are calculated across five-time durations listed in the following table. While the paper mention these durations as 100ms, 500ms, 1.5s, 10s, and 1min respectively, the abbreviations used in feature names introduce some doubt. The following table summarizes this. Nevertheless, we were able to reproduce the results of the original authors' paper with similar accuracy.

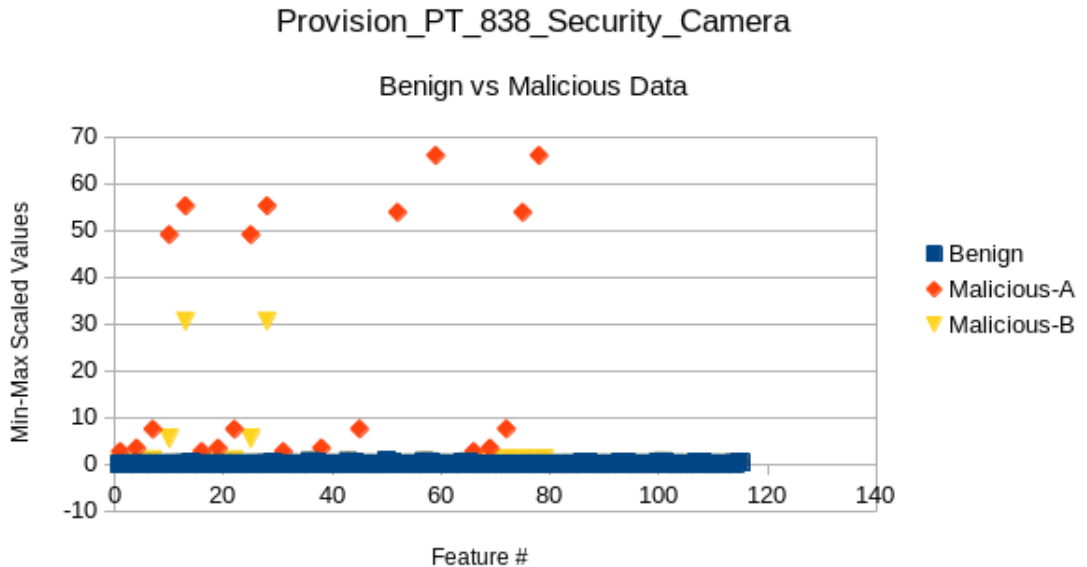


Figure 3.1: DDoS Classification

3.1.2 CIC-IDS2017 Dataset

The CIC-IDS2017 dataset [26] has been created by building a lab environment that simulates normal traffic originating from systems running common operating systems

such as Windows and Ubuntu. The dataset spans a collection period of 5 days, where the first day consists of only normal traffic, while the remaining days contain periods where various attacks were generated targeting the above-mentioned systems. The dataset consists of 85 features, extracted from the raw packet captures of the network traffic. The original dataset is arranged on a per-day basis. For the purpose of this thesis, we segregated the traffic records on the basis of the IP addresses of the attack victim machines. The following table lists the unique IP addresses, names, and the number of available records for each victim’s machine. We used the benign data records for “Monday” only.

The authors of the dataset have explained the process of dataset generation in their paper [12], where each attack was generated using publicly available tools on a Kali Linux machine. The following figure shows a scatter plot of sample data from the two servers (192.168.10.50 [Windows Server 16] 192.168.10.51 [Ubuntu Server 12]). The figure illustrates the stark difference in the values of some of the features belonging to the malicious dataset.

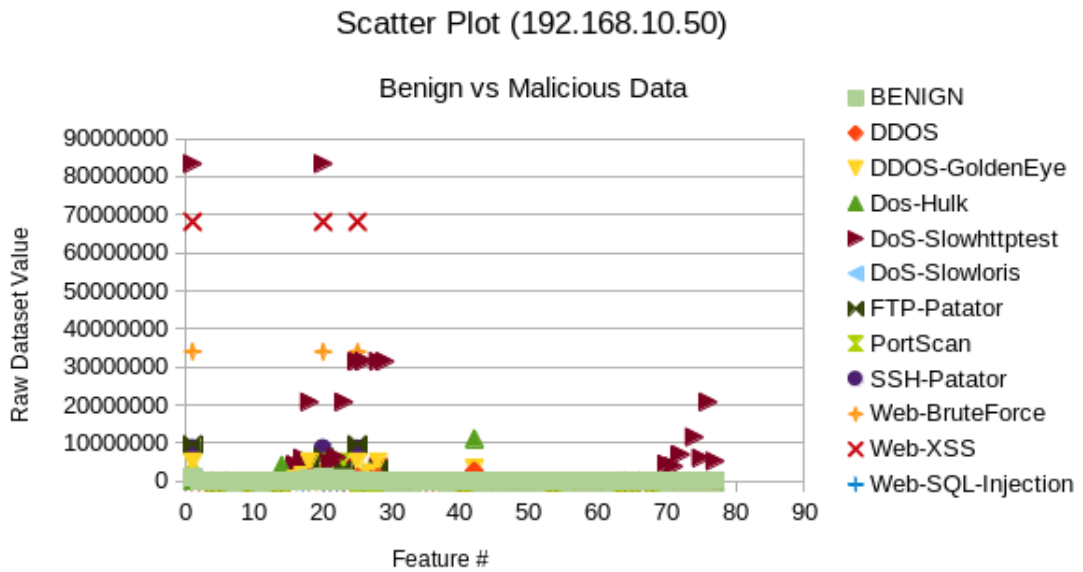


Figure 3.2: DDoS Classification

Name	IP Address	Record Category	Record Label	Record Count
Windows 8.1	192.168.10.5	Benign	Benign	62013
		Malicious	Bot	288
Windows VISTA	192.168.10.8	Benign	Benign	33033
		Malicious	Bot	748
			Infiltration	72
Windows 7 Pro	192.168.10.9	Benign	Benign	40468
		Malicious	Bot	372
Ubuntu 16.4	192.168.10.12	Benign	Benign	45289
		Malicious	Bot	2
Windows 10 Pro	192.168.10.14	Benign	Benign	38004
		Malicious	Bot	348
Windows 10	192.168.10.15	Benign	Benign	26671
		Malicious	Bot	580
Ubuntu 14.4	192.168.10.17	Benign	Benign	30302
		Malicious	Bot	7
Web Server 16	192.168.10.50	Benign	Benign	11340
		Malicious	DDoS	128027
			DoS-GoldenEye	10293
			DoS-Hulk	231073
			DoS-Slowhttptest	5499
			DoS-slowloris	5796
			FTP-Patator	7938
			PortScan	158930
			SSH-Patator	5897
			Web-Attack-Brute-Force	1507
			Web-Attack-Sql-Injection	21
	Web-Attack-XSS	652		
Ubuntu Server 12	192.168.10.51	Benign	Benign	28497
		Malicious	Heartblead	11

Table 3.5: CIC-IDS2017 Dataset size

3.1.3 Programming Language Libraries

Implementation of tests against the hypothesis of this work can be broadly divided into two parts;

- Compilation of datasets into the desired format and directory structure such that iterating through them programmatically is easier.
- Coding the anomaly model and transfer learning implementations.

Basic compilation and evaluation of the datasets were done using *Bash* and various CLI-based tools available in it. This included concatenation or splitting of datasets, evaluating dataset sizes, verifying the sanity of the dataset, and building a directory hierarchy. All datasets used were in the comma-separated-values (CSV) format. The prepared directory structure was then uploaded to the Google Drive account.

For the machine-learning and transfer-learning implementation, all code was written in *Python*. Google's *Colab* based *Jupyter* notebooks connected to a simple CPU-based runtime and formed the underlying infrastructure. The Google CoLab notebook was connected with the Google Drive account in order to access the datasets for training and writing out the performance results.

Preprocessing and Metric libraries for scaling, dataset splitting, and performance evaluation respectively were used from the *SciKitLearn* package. *Keras* was used to create the autoencoder neural network, and compile and fit it.

Matplotlib was used for the generation of Training/Optimization loss graphs and mean-squared-error distribution of the Test dataset.

Pandas was used to hold the imported CSV based datasets, while *Numpy* was used in order to aid in the intermediate steps of the data preprocessing.

Other generic libraries such as '*time*' and '*os*' etc. were used in order to interact with the directories containing the datasets and record training time etc.

Libraries:

1. *Pandas*
2. *Numpy*
3. *Keras*

4. ScikitLearn
5. Matplotlib

Tools:

1. Bash
2. Python
3. Google Colab Jupyter Notebook

3.2 NBaIoT – Anomaly Detection Transfer Learning

With the NBaIoT dataset, our prime objectives were the following;

1. Evaluating the performance of an existing autoencoder-based anomaly model on data belonging to other IoT devices with varying degrees of similarity.
2. Evaluating if specific layers of the autoencoders could be retrained with limited new data, and thereby be “*transferred*” to a different IoT device with an overlapping feature set.
3. And how does transfer-learning hold up when done across malware of similar nature.

The following pseudo-code details the workflow adopted in order to gather data against our above-mentioned objectives. A recursive loop was run which trained an anomaly model for a particular IoT device, then evaluated its performance against all other IoT devices in the dataset before and after *transfer learning*.

There were four iterations that covered four possible scenarios with the NBaIoT dataset. These are as follows;

- With a scope limited to the Mirai dataset only; i.e the original model was trained on the Mirai dataset of one device and then transferred to the Mirai dataset of the remaining devices.

- With a scope limited to the Bashlite dataset only; i.e the original model was trained on the Bashlite dataset of one device and then transferred to the Bashlite dataset of the remaining devices.
- With the original model trained on **Mirai** dataset of a device and then transferred to the **Bashlite** dataset of the remaining devices.
- With the original model trained on **Bashlite** dataset of a device and then transferred to the **Mirai** dataset of the remaining devices.

The following pseudo-code presents the sequence of steps performed at a high level. For the iterations three and four (listed above), the algorithm was slightly modified in steps 18 and 19 such that the dataset loaded for the unknown device was the opposite of what was loaded for the known device. The complete code can be found on the Github repository [34]

```

1 MALWARE_DATASET = [Mirai, Bashlite]
2 LIST_OF_IOT_DEVICES = [...]
3 for each DATASET in MALWARE_DATASET
4   for each IOT_DEVICE in LIST_OF_IOT_DEVICES
5     Define this device as the KNOWN_DEVICE
6       Load BENIGN_DATA
7       Load MALICIOUS_DATA
8       Split BENIGN_DATA into 60/20/20 split for BENIGN_TRAINING,
9         ↪ BENIGN_OPTIMIZATION, BENIGN_TEST
10      Fit Scaler on the BENIGN_TRAINING data
11      Use the learned function to scale BENIGN_OPTIMIZATION,
12        ↪ BENIGN_TEST, and MALICIOUS-DATAT datasets
13      Calculate FISHER_SCORE of features and select whether to
14        ↪ use all or limited features for training
15      Train an autoencoder model for the selected features using
16        ↪ BENIGN_TRAINING and BENIGN_OPTIMIZATION datasets
17      Use the BENIGN_TEST dataset to define a threshold for
18        ↪ RECONSTRUCTION_ERROR of the autoencoder. This shall
19        ↪ be the ANOMALY_THRESHOLD

```


14 Create a new dataset containing a random mix of
↳ MALICIOUS_DATA and BENIGN_TEST

15 Run this dataset through the autoencoder and use the
↳ RECONSTRUCTION_ERROR and ANOMALY_THRESHOLD to
↳ declare whether an instance is anomalous or not

16

17 Select a device from the remaining list of devices and
↳ define it as the UNKNOWN_DEVICE

18 Load BENIGN_DATASET of the UNKNOWN_DEVICE.

19 Load MALICIOUS_DATASET of the UNKNOWN_DEVICE

20 Scale both datasets using the scaling function learned
↳ earlier

21 Run the dataset against the anomaly model of KNOWN_DEVICE;
↳ record the performance impact if any

22

23 Extract randomly 10% of the records from the
↳ BENIGN_DATASET of UNKNOWN_DEVICE

24 Split BENIGN_DATASET into BENIGN_TRAINING,
↳ BENIGN_OPTIMIZATION, BENIGN_TEST in a 60/20/20
↳ fraction

25 Freeze all layers of the autoencoder model except the
↳ three innermost layers

26 Re-initialize the weights of the innermost layers

27 Run Train cycles on the innermost layers of the
↳ autoencoder only

28 Use the BENIGN_TEST subset of the data to calculate the
↳ RECONSTRUCTION_ERROR and redefine the
↳ ANOMALY_THRESHOLD of the transferred model

29 Create a new dataset containing a random mix of
↳ MALICIOUS_DATA and TEST_DATA

30 Record model performance

Details on the rationale for selected important steps are as follows;

Step-8:

A 60/20/20 split was chosen for training the autoencoder model. Since a majority of the devices had significantly fewer records for benign data as compared to the malicious data of Mirai and Bashlite, this split was chosen so that ample data could be presented to the autoencoder model for training and optimization (80% in total). Smaller subsets for optimization and testing might have led to insufficient model tuning or insufficiently randomized test space for performance evaluation of the anomaly model.

Step-9:

The Min-Max *Scaler* from the *ScikitLearn* library was used to scale the training data. The scaling function is learned only on the original KNOWN-DEVICE'S training data subset. The remaining subsets (Optimization, Test, and Malicious) used this learned scaling function to normalize the dataset.

Step-11:

The Fisher score was used as a means of evaluating how the model performance would be impacted by using only the top features with the highest degree of in. Our iterations included runs consisting of all features as well as with top features selected on the basis of their Fisher scores only. For the latter, a feature was selected only if the Fisher score was larger than 0.1. In the runs that used selected feature sets, the consequent transfer learning used the exact same set of features from the UNKNOWN-DEVICE'S dataset as well.

Step-12:

The autoencoder model consists of Dense layers, with each layer's size linearly decreasing until the encoder latent representation is 20% of the size of the selected number of features. The following diagram presents the layout of the autoencoder model used.

Following model training parameters were used;

Loss Function: Mean Squared Error

Optimizer: Adam. Adam optimization is a stochastic gradient descent method that is based on the adaptive estimation of first-order and second-order moments. According to Kingma et al [15], the method is "computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems

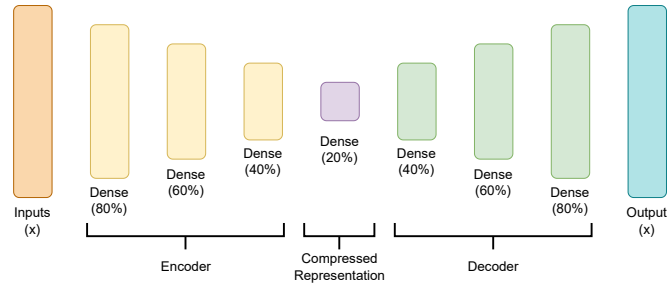


Figure 3.3: Autoencoder Pre-Transfer Learning

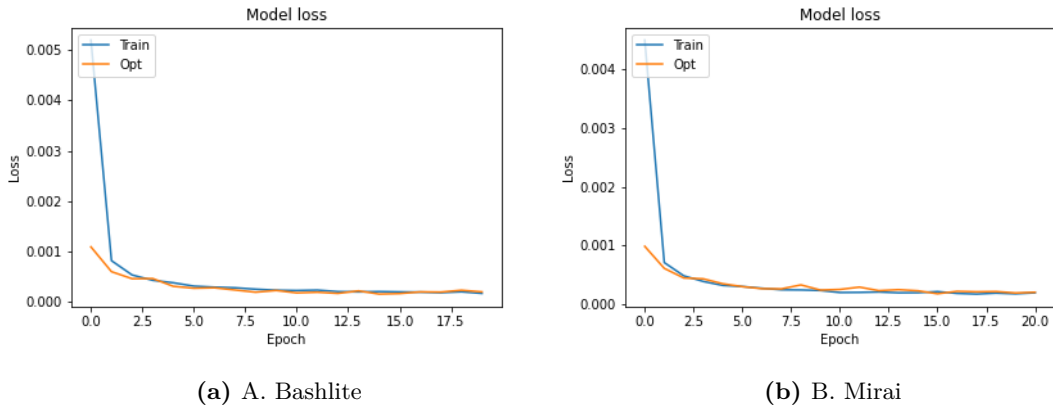


Figure 3.4: Loss Function plot of the IoT device SimpleHome-XCS7-1003-WHT-Security-Camera for Bashlite and Mira datasets respectively

that are large in terms of data/parameters"

Max Epochs: 100

EarlyStopping Patience: 5, If the values of the loss function do not improve for five consecutive epochs, stopping the model fitting process.

The autoencoder model is trained with an aim to minimize the reconstruction error of “benign” data records. We calculate the mean-squared loss between the reconstructed output x^R and the original input x for all records in our “Test” subset of the data. The maximum values of the mean-squared error form the upper boundary of the anomaly threshold. However, it is entirely possible that the record with the maximum reconstruction error might itself be an anomalous entry. In order to cater to this possibility, we extract a list of candidate thresholds by siphoning off the highest mean-squared error values by percentile. Mean-squared errors in the percentile range of 90 to 100 are evaluated with increments of 0.05 each. The one that maximizes the accuracy of the

model is selected.

$$\sum_{i=1}^D (x_i - x_i^R)^2 \quad (3.2.1)$$

Step-17 – 21: Once a model for a particular device is trained, its performance is tested for datasets of all the remaining IoT devices. In a vast majority of cases, the accuracy of the model decreased.

Step-24: In order to simulate the case that transfer learning is usually done with a very limited set of somewhat similar data, we use only 10% of the total benign dataset records. This sample is further split into 60/20/20 portions for model training, optimization, and testing. The following table illustrates the number of records used from each IoT device’s dataset when used as a KNOWN-DEVICE and when used as an UNKNOWN-DEVICE to which an autoencoder is “transfer-learned”.

Table 3.6: Representation of the Dataset size and its consumption during training stages

Device	Abbreviation	Total Data	Original Model Training			Transfer-Learning		
			Training	Optimization	Validation	Training	Optimization	Validation
Danmini Doorbell	DAD	49,548	29,729	9,910	9,910	2,973	991	991
Ecobee Thermostat	ECT	13,113	7,868	2,623	2,623	787	262	262
Ennio Doorbell	END	39,100	23,460	7,820	7,820	2,346	782	782
Philips B120N10 Baby Monitor	PBM	175,240	105,144	35,048	35,048	10,514	3,505	3,505
Provision PT 737E Security Camera	P737	62,150	37,290	12,430	12,430	3,729	1,243	1,243
Provision PT 838 Security Camera	P838	98,514	59,108	19,703	19,703	5,911	1,970	1,970
Samsung SNH 1011 N Webcam	SNH	52,150	31,290	10,430	10,430	3,129	1,043	1,043
SimpleHome XCS7 1002 WHT Security Camera	S1002	46,581	27,949	9,316	9,316	2,795	932	932
SimpleHome XCS7 1003 WHT Security Camera	S1003	19,528	11,717	3,906	3,906	1,172	391	391

Step-25: In over case of transfer learning, the innermost three layers of the autoencoder are replaced with freshly initialized values, implying that these layers have no memory of being trained in the previous steps. Keras by default uses GloRotUniform for initializing model weights which were used for the re-initialization as well. The following code shows how this was done. Further, when retraining the model, all layers were frozen except the innermost three layers. This was in conjunction with the transfer learning concept of only retraining the newly introduced layers.

The following code snippet represents how the innermost layers of the autoencoder are *reset*, as a way of simulating transfer learning.

```

1 initializer = keras.initializers.GlorotUniform()
2 for layer in transfer_model.layers[3:6]:
3     layer.set_weights([initializer(shape=w.shape) for w in layer.
        ↪ get_weights()])

```

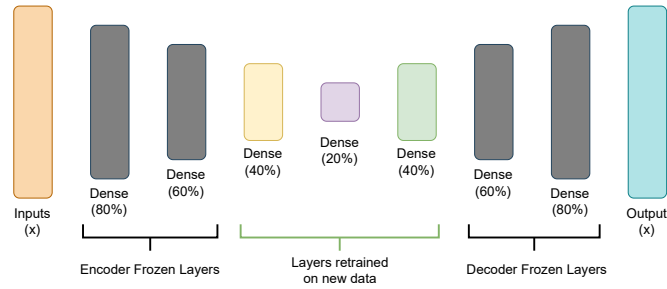


Figure 3.5: Autoencoder Post-Transfer Learning

Step-28: Now that our autoencoder model has changed, the reconstruction error of the model will have changed as well. We re-calculate the threshold value for our anomaly detection engine as a percentile of the mean-squared reconstruction errors. We evaluated that the highest accuracy resulted if the error threshold was kept above the 90th percentile.

3.3 CIC-IDS2017 – Anomaly Detection

We decided to include the CIC-IDS2017 dataset in order to cover a deficiency of the NBaIoT dataset, i.e its lacking similarity with real-world IP traffic flows, for which standards such as the IPFIX and NetFlow-v9 / NetFlow-v10 are used.

While the autoencoder-based anomaly model was able to deliver exceptional results on the NBaIoT dataset, the dataset consisted of unconventional feature attributes that would require custom data preprocessors in a real-world scenario. With this dataset, our prime objective was to evaluate the performance of autoencoder-based anomaly models on traffic flow information which is much more similar to the kind of network flows available with ISPs and enterprises.

A majority of the CIC-IDS2017 dataset catered to Bot activities such as data theft and email spamming, however, the dataset also contains malicious traffic records comprising

a variety of DoS attacks, Web-based intrusion attempts, and port-scanning. However, since this dataset only contained a single device against which the above-mentioned attacks were simulated, transfer learning evaluations could not be made. Nevertheless, we were able to train an anomaly model on the network flow data and observed a range of performance accuracy, ranging from a mere 69% to over 90% for 6 of the 11 attacks. The autoencoder-based anomaly model performed significantly better on DoS and Web-Intrusion attempts as compared to Bot communication.

The following pseudo-code presents the workflow of anomaly detection in the CIC-IDS2017 dataset.

```

1  DICTIONARY-OF-DEVICE-IDS-DATA-FILES = {...}
2  for each DEVICE in DICTIONARY-OF-DEVICE-IDS-DATA-FILES
3      for each IDSCATEGORY against DEVICE
4          Load BENIGN-DATA
5          Load MALICIOUS-DATA
6          Data Preprocessing/Cleaning (e.g Fill any empty cells or
           ↪ NaN with value 0)
7          Split BENIGN-DATA into 60/20/20 split for BENIGN-TRAINING,
           ↪ BENIGN-OPTIMIZATION, BENIGN-TEST
8          Fit Scaler of BENIGN-TRAINING dataset and use the learned
           ↪ function to scale remaining BENIGN-OPTIMIZATION,
           ↪ BENIGN-TEST, and MALICIOUS-DATA datasets
9          Calculate the FISHERSCORE of features and select whether
           ↪ to use all or limited features for training.
10         Train an autoencoder model for the selected features
11         Use the BENIGN-TEST portion of the dataset to define a
           ↪ threshold for RECONSTRUCTION-ERROR of the
           ↪ autoencoder. This shall be the ANOMALY-THRESHOLD.
12         Create a new dataset containing a random mix of MALICIOUS-
           ↪ DATA and BENIGN-TEST datasets.
13         Run this dataset through the autoencoder and use the
           ↪ RECONSTRUCTION-ERROR and ANOMALY-THRESHOLD defined
           ↪ earlier to declare anomaly if any.

```

The anomaly detection iterations on the CIC-IDS2017 dataset were similar to those performed on the NBaIoT dataset. This dataset required a few steps for data preparation such as filling in empty columns with “0” values.

Multiple code runs were made using a variety of optimizer algorithms

Loss Function: Mean Squared Error

Optimizer: A variety of optimization algorithms were tried out, however, we found “Adam” and “Adamax” to be the better performers.

Max Epochs: 100

EarlyStopping Patience: 5, If the values of the loss function do not improve for five consecutive epochs, stopping the model fitting process.

CHAPTER 4

Results and Discussion

Chapter Summary

We delve into the details of the model performance for each experimental iteration, present the process of deriving the performance parameters and comment on the results.

A multitude of iterations was run for each dataset. A confusion matrix was built for each run of the anomaly model. Performance metrics of Accuracy, Precision, Recall, and F1-Score were extracted from the confusion matrix and tabulated in order to build a visual representation of the anomaly models' capability. Additionally, the model training times were noted as well.

A brief description of each of these metrics is as follows;

True positives (TP): When the model predicts that an observation belongs to a class and it actually does belong to that class.

True negatives (TN): When the model predicts that observation does not belong to a class and it actually does not belong to that class.

False positives (FP): When the model predicts that an observation belongs to a class when in reality it does not.

False negatives (FN): When the model predicts that observation does not belong to a class when in fact it does.

Accuracy: It is the percentage of correct predictions for the test data and calculated as follows;

$$\frac{(TP + TN)}{(TP + TN + FP + FN)} \quad (4.0.1)$$

Precision: It is the fraction of relevant examples among all of the examples which were predicted to belong in a certain class. Precision answers the question "How many of the selected items are relevant?"

$$\frac{TP}{(TP + FP)} \quad (4.0.2)$$

Recall: It is the fraction of examples that were predicted to belong to a class with respect to all of the examples that truly belong in the class. Recall answers the question "How many of the relevant items were selected?"

$$\frac{TP}{(TP + FN)} \quad (4.0.3)$$

F1-Score: The F1 score is a good measure of summarizing the model's accuracy and recall into a single value. F1-Score gives an equal bias to both accuracy and precision,

hence the value of beta is equal to 1.

$$F\beta = \frac{(1 + \beta^2)precision \cdot recall}{(\beta^2)precision + recall} \quad (4.0.4)$$

$$F1 = \frac{precision \cdot recall}{precision + recall} \quad (4.0.5)$$

4.1 NBaIoT Dataset - Anomaly Detection and Transfer Learning

4.1.1 Evaluating Model Accuracy

For the NBaIoT dataset, a matrix of model accuracy was built as a performance comparison before and after transfer learning. A total of 520 iterations of the anomaly model were run across both categories (Mirai & Bashlite), with and without the feature selection process based on Fisher-Score. Within these iterations, new autoencoder models were trained 32 times and transfer-learned 228 times. At each recursion, metrics of the confusion matrix, F-1 score, and training time were collected. The accuracy matrices presented in Figure 4.1, 4.2, 4.3 and 4.4 aim to build a visual representation of the anomaly model’s performance for each iteration on the NBaIoT dataset. Table 4.1 briefly explains the experimental iteration that produced the respective accuracy matrices in the aforementioned figures.

Table 4.1: Experimental scope against each accuracy matrices

Figure	Explanation
4.1 MIRAI Transfer Learning Results	The scope is limited to the Mirai dataset of NBaIoT. An anomaly model is trained on an IoT device and then tested against the dataset of all remaining IoT devices before and after transfer-learning
4.2 BASHLITE Transfer Learning Results	The scope is limited to the Bashlite dataset of NBaIoT. An anomaly model is trained on an IoT device and then tested against the dataset of all remaining IoT devices before and after transfer-learning
4.3 MIRAI to BASHLITE Transfer Learning Results	An anomaly model is trained on the Mirai dataset of IoT devices and then tested against the Bashlite dataset of all remaining IoT devices before and after transfer-learning
4.4 BASHLITE to MIRAI Transfer Learning Results	An anomaly model is trained on the Bashlite dataset of IoT devices and then tested against the Mirai dataset of all remaining IoT devices before and after transfer-learning

For each matrix, the devices listed in the left-most column are the original “**Known**” device on which the anomaly model was trained. The devices listed in the last row are the “**Unknown**” devices of varying similarity. The counter diagonal of the matrix is the accuracy of the model on the same device’s dataset it was trained on, while all other cells represent the model’s accuracy on the dataset of the “**Unknown**” device. For example, the first cell containing an accuracy of 58.306% corresponds to a model

CHAPTER 4: RESULTS AND DISCUSSION

trained on *Danmini Doorbell* and tested against the dataset of *Philips Baby Monitor* represents the model performance before transfer-learning. Post-transfer-learning, this value increases to 99.984%.

Original Device (on which model is trained)	Category	Name	Accuracy Percentage										
Original Device (on which model is trained)	DoorBell	DAD	58.306%	86.088%	91.752%	71.023%	84.586%	95.247%	99.997%		Before Transfer Learning		
	Thermostat	ECT	50.060%	83.963%	91.496%	71.409%	82.293%	99.987%	87.848%				
	Security Camera	P737	94.265%	91.638%	99.949%	99.964%	99.995%	99.580%	99.990%				
		P838	94.262%	91.584%	99.368%	99.995%	99.992%	99.771%	99.960%				
		S1003	90.547%	89.974%	99.987%	76.155%	91.207%	99.314%	99.990%				
		S1002	91.977%	99.996%	99.846%	92.579%	93.403%	98.360%	100.000%				
	Baby Monitor	PBM	99.997%	93.613%	99.923%	97.274%	96.653%	99.962%	100.000%				
	Original Device (on which model is trained)	DoorBell	DAD	99.984%	96.061%	99.166%	100%	99.960%	99.860%	99.997%			After Transfer Learning
		Thermostat	ECT	99.984%	96.583%	98.495%	96.836%	99.938%	99.987%	99.841%			
		Security Camera	P737	99.995%	99.200%	99.656%	96.959%	99.995%	99.843%	99.419%			
P838			99.984%	99.834%	99.923%	99.995%	99.927%	99.775%	99.686%				
S1003			99.947%	98.770%	99.987%	95.839%	99.947%	99.784%	99.471%				
S1002			99.991%	99.996%	99.616%	98.991%	99.900%	99.822%	99.412%				
Baby Monitor		PBM	99.997%	99.329%	99.565%	98.146%	99.904%	99.297%	99.490%				
			PBM	S1002	S1003	PT838	PT737	ECT	DAD				
			B. Monitor	Security Camera				Thermostat	DoorBell				
Device on whose data the model's performance is tested on													

Figure 4.1: MIRAI Transfer Learning Results

Original Device (on which model is trained)	Category	Name	Accuracy Percentage										
Original Device (on which model is trained)	DoorBell	DAD	90.716%	50.010%	50.034%	50.032%	50.000%	64.567%	73.065%	50.000%	99.743%		Before Transfer Learning
	Thermostat	ECT	54.731%	64.190%	50.009%	53.456%	64.319%	50.822%	50.732%	98.971%	50.020%		
	Security Camera	P737	96.522%	51.649%	60.625%	50.075%	50.026%	80.875%	99.735%	50.229%	99.788%		
		P838	97.558%	51.649%	92.196%	50.000%	50.128%	99.492%	50.000%	50.000%	99.627%		
		S1003	54.923%	58.150%	94.236%	95.524%	98.989%	65.602%	55.020%	50.953%	50.091%		
		S1002	56.547%	50.019%	52.568%	99.238%	97.080%	67.668%	81.295%	59.077%	50.061%		
	Baby Monitor	PBM	99.591%	97.124%	99.703%	95.116%	98.822%	88.884%	90.000%	99.695%	99.586%		
	Webcam	SNH	93.734%	98.993%	60.280%	94.472%	67.008%	78.190%	78.190%	91.419%	93.864%		
	DoorBell	END	98.996%	51.707%	50.020%	49.989%	50.026%	69.424%	74.336%	53.242%	99.606%		
	Original Device (on which model is trained)	DoorBell	DD	97.575%	99.574%	98.389%	96.762%	96.973%	99.697%	99.632%	99.373%	99.743%	
Thermostat		ECT	98.134%	99.693%	98.690%	68.225%	95.226%	99.147%	99.394%	98.971%	99.714%		
Security Camera		P737	98.713%	99.631%	99.422%	98.246%	98.253%	98.550%	99.735%	99.856%	99.659%		
		P838	97.756%	97.484%	99.744%	98.620%	97.790%	99.492%	97.519%	99.343%	99.748%		
		S1003	97.133%	99.185%	99.617%	99.443%	98.989%	99.617%	98.918%	99.242%	99.878%		
		S1002	99.244%	99.136%	52.568%	99.238%	98.364%	99.424%	98.920%	98.305%	99.637%		
Baby Monitor		PBM	99.031%	83.513%	99.703%	84.442%	83.825%	99.133%	98.132%	86.138%	99.645%		
Webcam		SNH	99.679%	98.993%	99.651%	96.468%	99.266%	99.398%	99.398%	99.356%	99.716%		
DoorBell		END	98.996%	98.841%	99.047%	99.389%	97.747%	99.675%	98.922%	98.462%	99.840%		
			END	SNH	PBM	S1002	S1003	PT838	PT737	ECT	DAD		
		DoorBell	Webcam	B. Monitor	Security Camera				Thermostat	DoorBell			
Device on whose data the model's performance is tested on													

Figure 4.2: BASHLITE Transfer Learning Results

CHAPTER 4: RESULTS AND DISCUSSION

Original Device's MIRAI dataset (on which model is trained)	Category	Name	Accuracy Percentage									
	Original Device's MIRAI dataset (on which model is trained)	DoorBell	DAD	97.033%	57.450%	50.026%	49.968%	63.704%	54.573%	91.593%	58.314%	99.995%
Thermostat		ECT	97.494%	37.507%	50.037%	64.341%	66.265%	56.497%	98.463%	99.981%	99.808%	
Security Camera		P737	67.238%	64.861%	65.547%	64.706%	64.882%	77.393%	99.984%	60.107%	68.803%	
		P838	67.596%	65.053%	66.857%	65.854%	67.572%	99.995%	89.220%	66.323%	69.409%	
		S1003	65.908%	63.730%	96.248%	66.423%	99.987%	49.259%	68.600%	54.424%	68.480%	
Baby Monitor		S1002	66.049%	74.803%	64.349%	99.962%	68.007%	52.314%	68.391%	33.333%	68.843%	
		PBM	67.174%	98.917%	99.990%	65.522%	67.162%	65.009%	68.190%	67.124%	68.621%	
Original Device's BASHLITE dataset (on which model is trained)		DoorBell	DAD	98.964%	98.390%	97.441%	95.037%	95.804%	68.499%	97.730%	67.404%	99.995%
		Thermostat	ECT	97.786%	99.692%	98.402%	95.537%	98.543%	99.288%	99.280%	99.981%	99.698%
		Security Camera	P737	95.234%	97.718%	99.262%	98.321%	98.575%	99.475%	99.984%	97.369%	99.684%
	P838		99.179%	99.508%	99.562%	98.438%	99.343%	99.995%	97.890%	67.501%	99.618%	
	S1003		98.586%	99.015%	99.049%	96.192%	99.987%	99.391%	99.714%	67.294%	99.710%	
	Baby Monitor	S1002	98.225%	97.744%	99.096%	99.962%	96.057%	99.067%	99.314%	99.598%	99.638%	
		PBM	97.987%	97.555%	99.990%	95.446%	96.447%	95.000%	98.716%	99.509%	99.617%	
		END	SNH	PBM	S1002	S1003	PT838	PT737	ECT	DAD		
		DoorBell	Webcam	B. Monitor	Security Camera			Thermostat	DoorBell			

Device on whose BASHLITE dataset the model's performance is tested on and transferred to

Figure 4.3: MIRAI to BASHLITE Transfer Learning Results

Original Device's BASHLITE dataset (on which model is trained)	Category	Name	Accuracy Percentage								
	Original Device's BASHLITE dataset (on which model is trained)	DoorBell	DD	No Mirai Dataset for these devices	50.000%	53.617%	50.000%	50.000%	50.000%	50.000%	50.000%
Thermostat		ECT	50.000%		72.617%	67.623%	50.000%	74.602%	99.218%	85.446%	
Security Camera		P737	50.000%		73.680%	50.000%	50.000%	97.744%	50.000%	94.661%	
		P838	50.011%		87.559%	50.282%	99.246%	50.000%	53.242%	99.435%	
		S1003	50.006%		86.518%	97.990%	50.000%	89.123%	53.204%	99.495%	
Baby Monitor		S1002	50.964%		99.936%	96.081%	56.862%	99.083%	53.089%	99.627%	
		PBM	98.984%		96.018%	68.571%	84.347%	99.799%	66.438%	99.889%	
Webcam		SNH	98.226%		50.009%	88.074%	96.004%	50.000%	98.359%	53.204%	99.546%
DoorBell		END	96.240%		50.000%	50.000%	50.026%	50.000%	50.000%	50.000%	50.000%
Original Device's MIRAI dataset (on which model is trained)		DoorBell	DD		No Mirai Dataset for these devices	99.989%	99.626%	99.687%	99.969%	99.944%	99.708%
	Thermostat	ECT	99.974%	99.909%		99.650%	99.906%	99.842%	99.218%	99.745%	
	Security Camera	P737	99.981%	99.816%		99.903%	99.933%	97.744%	99.208%	99.973%	
		P838	99.977%	99.864%		99.568%	99.246%	99.881%	99.716%	99.959%	
		S1003	99.981%	99.936%		97.990%	99.959%	99.956%	99.712%	99.873%	
	Baby Monitor	S1002	99.981%	99.936%		97.990%	99.959%	99.956%	99.712%	99.873%	
		PBM	98.984%	99.940%		99.912%	99.878%	99.836%	99.424%	99.978%	
	Webcam	SNH	98.226%	99.945%		99.708%	99.644%	99.973%	99.977%	99.229%	99.845%
	DoorBell	END	96.240%	98.871%		99.696%	99.093%	51.277%	65.616%	89.442%	99.924%
		END	SNH	PBM		S1002	S1003	PT838	PT737	ECT	DAD
	DoorBell	Webcam	B. Monitor	Security Camera			Thermostat	DoorBell			

Device on whose MIRAI dataset the model's performance is tested on and transferred to

Figure 4.4: BASHLITE to MIRAI Transfer Learning Results

In general, a significant decrease in the model’s accuracy is noticed when it is tested on a different “**Unknown**” IoT device’s data. However, this decrease in model accuracy is recovered after the process of transfer-learning and re-defining the anomaly threshold. We observed that keeping the anomaly threshold equal to the maximum value of mean-squared-error of the test dataset started introducing False-Positives; this was investigated by plotting the mean-squared-error graphs of “benign” data as well as “malicious” data. The minimum and maximum values were found to be overlapping slightly. This overlap was the root cause of false positives. In order to maximize performance, the threshold definition had to be such that it minimized the impact of this overlap. A slightly lower threshold based on the percentile value was found to give a significant improvement in the anomaly model.

From the model accuracy matrices presented in the figures 4.1, 4.2, 4.3 and 4.4, the “**percentage-accuracy-decrease**” was calculated for each model trained on an IoT device and tested on other devices before and after transfer learning. The following formula was used.

$$\%AccuracyDecrease = \frac{ModelAccuracy_{KnownDevice} - \sum \frac{ModelAccuracy_{UnknownDevices}}{\#UnknownDevices}}{ModelAccuracy_{KnownDevice}} \cdot 100 \quad (4.1.1)$$

Figure 4.1 presents the accuracy matrix of model accuracy across IoT devices within the Mirai dataset. As shown in Table 4.2, the average detection accuracy of an anomaly model decreased by 8.68% when tested against an “**Unknown**” device pre-transfer learning. This reduced to an average of 0.75% post-transfer learning. Anomaly models trained on the Mirai dataset of the Ecobee Thermostat posted the highest impact on accuracy at 22.15% pre-transfer learning and 1.37% post-transfer learning.

Figure 4.2 presents the accuracy matrix of model accuracy across IoT devices within the Bashlite dataset. As compared to the Mirai dataset, the average accuracy decrease on the Bashlite dataset was much higher, with an average of 30.64% pre-transfer learning and 2.33% post-transfer learning. In this iteration too, the Ecobee Thermostat posted the highest impact on model accuracy at 44.65% and 4.24% post-transfer learning. Details can be perused in Table 4.3.

Figure 4.3 and 4.4 are accuracy matrices corresponding to the third and fourth iteration of the experiment. These consist of instances where an anomaly model trained on the

Mirai dataset of an IoT device was tested against the *Bashlite* dataset of all other devices before & after transfer-learning and vice versa. Table 4.3 summarises the percentage decrease in model accuracy before and after transfer-learning.

Table 4.2: Percentage decrease in model accuracy before and after transfer-learning (TL) across device

IoT Device	Mirai (% decrease)		Bashlite (% decrease)	
	Pre-TL	Post-TL	Pre-TL	Post-TL
DAD	18.83	0.82	40.04	1.25
ECT	22.15	1.37	44.65	4.24
P737	2.43	0.82	32.35	0.70
P838	2.51	0.14	32.01	1.00
S1003	8.79	1.03	33.77	-0.14
S1002	3.97	0.37	35.22	6.08
PBM	2.09	0.71	3.61	7.99
SNH	-	-	17.02	-0.12
END	-	-	37.07	0.01
Average	8.68	0.75	30.64	2.33

Table 4.3: Percentage decrease in model accuracy before and after transfer-learning (TL) across device and malware type

IoT Device	MIRAI to BASHLITE (% decrease)		BASHLITE to MIRAI (% decrease)	
	Pre-TL	Post-TL	Pre-TL	Post-TL
DAD	34.66	10.09	49.14	-0.33
ECT	28.68	1.45	32.76	-0.62
P737	33.30	1.78	37.19	-2.11
P838	30.26	4.87	34.42	-0.59
S1003	33.36	5.12	27.14	-1.95
S1002	37.97	1.37	24.00	0.36
PBM	29.03	2.46	13.28	-0.85
SNH	-	-	22.16	-1.56
END	-	-	48.04	10.36
Average (%)	32.47	3.88	32.02	0.30

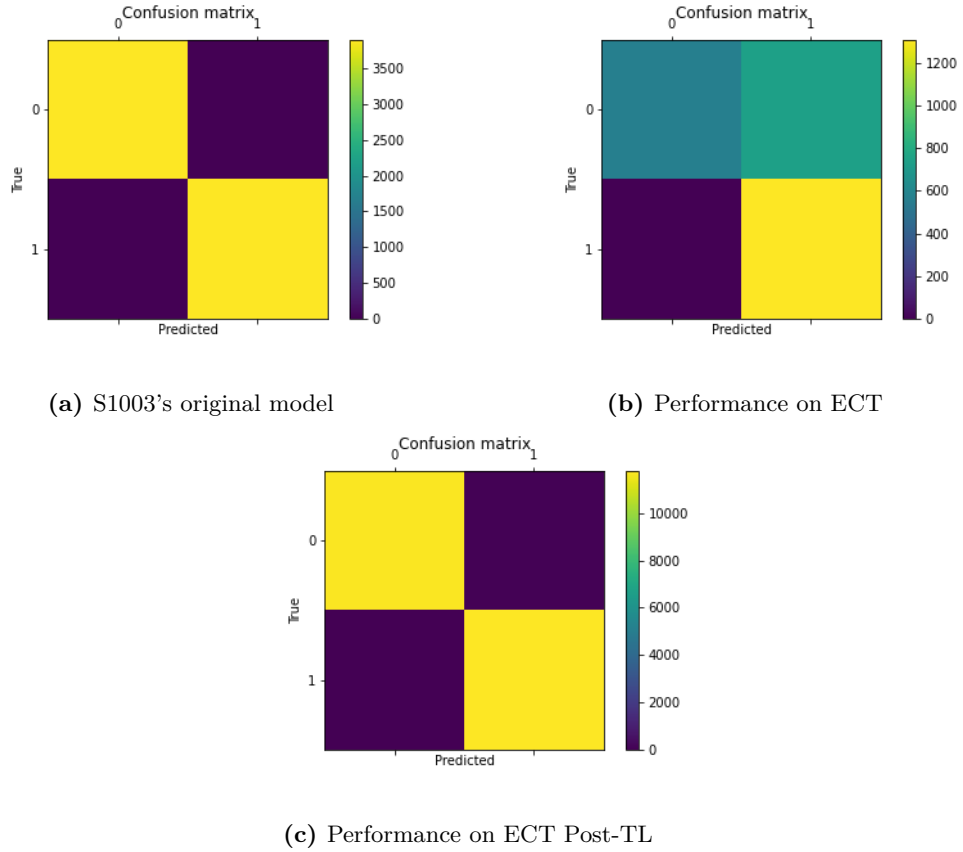


Figure 4.5: Anomaly Detector’s performance change for MIRAI dataset

Figure 4.5 consists of three confusion matrices and presents an example of how the performance of the anomaly model varied through the three stages; i.e model performance on the same device upon which the model was trained, model performance on a different device’s data before transfer-learning, and model performance on a different device’s data after transfer learning.

4.1.2 Evaluating Model Training Time

Another core aspect of the experimentation included evaluating the amount of time saved when an anomaly model for an IoT device is transfer-learned as opposed to trained from scratch.

Model training times were thus noted for each cycle of training and transfer-learning against the IoT device. These were populated into a matrix similar in structure to the ones made for model accuracy. Figure 4.6 and 4.7 represent the amount of time taken to train a model from scratch (denoted by the counter diagonal of the matrix), and to

transfer learning onto a different IoT device. These figures correspond to the first two iterations of the experiment conducted on the Mirai and Bashlite datasets respectively. Table 4.4 and 4.7 summarize the percentage decrease in the time required to train an anomaly model by transfer-learning as opposed to training from scratch. The Mirai dataset averaged 47.31% while the Bashlite dataset saved 58.27% of the time when the anomaly model for an IoT device was transfer-learned instead of being learned from scratch. This capability of having the time required to train the anomaly model can have huge benefits in large-scale production environments.

$$\%TimeSaved = \frac{\sum \frac{TransferLearningTime_{knowndevice}}{\#ofinstances} - TrainingTime_{knowndevice}}{TrainingTime_{knowndevice}} \cdot 100 \quad (4.1.2)$$

Table 4.4: Time saved when transfer-learning on Mirai dataset

IoT Device	Average Training Time (seconds)		% decrease
	Original Model	Transfer Learning	
DAD	118.7	67.0	43.50%
ECT	48.5	25.9	46.65%
P737	77.9	50.0	35.79%
P838	159.3	102.5	35.67%
S1003	34.0	25.0	26.47%
S1002	100.9	30.3	70.01%
PBM	348.9	93.8	73.12%
Average	126.9	56.4	47.31%

Table 4.5: Time saved when transfer-learning on Bashlite dataset

IoT Device	Average Training Time (seconds)		% decrease
	Original Model	Transfer Learning	
DAD	120.5	59.1	50.94%
ECT	104.6	30.6	70.78%
P737	149.5	93.6	37.38%
P838	491.4	133.1	72.92%
S1003	60.4	36.0	40.32%
S1002	257.0	49.0	80.92%
PBM	776.7	126.5	83.72%
SNH	205.2	67.2	67.26%
END	83.7	66.8	20.23%
Average	249.9	73.6	58.27%

Original Device (on which model is trained)	Category	IoT Device	Training Time (seconds)								
Security Camera	DoorBell	DAD	136.0	44.1	27.3	139.1	46.2	36.6	118.7		
	Thermostat	ECT	139.9	24.9	13.8	141.3	61.8	48.5	82.2		
	Baby Monitor	P737	71.9	43.6	25.1	113.9	77.9	41.2	58.8		
		P838	84.4	21.7	29.0	159.3	68.4	23.5	56.6		
		S1002	77.1	20.2	34.0	80.9	41.9	16.0	82.2		
		S1003	53.6	100.9	16.8	52.5	34.6	13.0	82.2		
Baby Monitor	PBM	348.9	27.1	38.1	87.0	47.3	24.9	40.3			
		B. Monitor	Security Camera					Thermostat	DoorBell		
		PBM	S1002	S1003	P838	P737	ECT	DAD			

Device on whose data the model's performance is tested on

Figure 4.6: Model Training times for Mirai dataset

Original Device (on which model is trained)	Category	IoT Device	Training Time (seconds)										
DoorBell	DAD	82.3	90.9	255.6	142.2	33.0	170.4	106.3	33.7	120.5			
	Thermostat	ECT	69.2	64.9	154.8	39.1	24.1	120.1	101.8	104.6	77.7		
Security Camera	P737	82.2	142.2	48.3	49.5	21.7	153.5	149.5	41.3	47.0			
	P838	44.0	92.6	103.3	51.3	36.8	491.4	142.2	41.6	61.1			
	S1002	59.5	40.5	213.6	24.6	60.4	136.5	64.5	41.3	74.1			
	S1003	65.8	57.7	87.7	257.0	25.3	95.8	68.1	17.6	70.9			
Baby Monitor	PBM	48.8	32.7	776.7	43.4	41.4	122.2	48.6	29.0	24.0			
Webcam	SNH	82.3	205.2	70.6	33.2	23.8	202.3	75.3	29.2	80.2			
DoorBell	END	83.7	16.1	78.1	55.9	82.2	202.2	142.2	10.9	42.6			
		DoorBell	Webcam	B. Monitor	Security Camera					Thermostat	DoorBell		
		END	SNH	PBM	S1002	S1003	P838	P737	ECT	DAD			

Device on whose data the model's performance is tested on

Figure 4.7: Model Training times for Bashlite dataset

4.1.3 Correlating Static Features with Model Performance

Table 4.6 lists the static feature sets of the IoT devices. This table was created based on the listed attributes of the IoT product with the intention of finding any possible correlation between them and the transfer learning performance. Figure reffig:device-feature-percentage-overlap reflects the IoT device’s features as a "percentage-overlay". This matrix was made in order to build an intuitive sense of any co-relation between the static feature set of IoT devices and the transferability of their anomaly model.

Table 4.6: IoT Device Feature Distribution in NBaIoT dataset

Device	Total Features	Network			Camera		Audio			Display	Sensor					SD Card	
		WiFi	LAN	ZigBee	Picture	Video	In	TwoWay	Record		Speaker	Thermal	Proximity	Motion	Humidity		Noise
DAD	8	1	1	0	1	1	1	1	0	1	0	0	0	1	0	0	0
END	8	1	0	0	1	1	1	1	0	1	1	0	0	1	0	0	0
ECT	7	1	0	1	0	0	0	0	0	1	1	1	1	1	0	0	0
P737	7	1	1	0	0	1	1	1	0	0	0	0	0	1	0	0	1
P838	7	1	1	0	0	1	1	1	0	0	0	0	0	1	0	0	1
S1003	4	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1
S1002	7	1	0	0	0	1	1	1	0	1	0	0	0	1	0	0	1
PBM	9	1	0	0	0	1	1	1	0	1	0	1	0	1	1	1	0
SNH	4	1	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0

A few results pointed towards a possibility of correlation; e.g the Philips Baby Monitor had the most static features and its anomaly model was the highest performing in all four iterations. However, other results seemed to contradict this observation; e.g

Category	Device Name	No. of Static features	Percentage Overlap in Static Features								
DoorBell	DAD	8	100.00%	75.00%	100.00%	100.00%	100.00%	85.71%	28.57%	87.50%	-
	END	8	100.00%	75.00%	85.71%	75.00%	71.43%	62.50%	42.86%	-	-
Thermostat	ECT	7	50.00%	57.14%	28.57%	50.00%	28.57%	25.00%	-	-	-
Security Camera	PT37	7	100.00%	71.43%	85.71%	100.00%	100.00%	-	-	-	-
	P838	7	100.00%	71.43%	85.71%	100.00%	-	-	-	-	-
	S1003	4	75.00%	75.00%	100.00%	-	-	-	-	-	-
	S1002	7	75.00%	85.71%	-	-	-	-	-	-	-
Baby Monitor	PBM	9	75.00%	-	-	-	-	-	-	-	-
Webcam	SNH	4	-	-	-	-	-	-	-	-	-
			4	9	7	4	7	7	7	8	8
			SNH	PBM	S1002	S1003	PT838	PT737	ECT	END	DAD
			Webcam	Baby Monitor	Security Camera			Thermostat	DoorBell		

Figure 4.8: Percentage Feature Overlap of IoT devices

the anomaly model of Simple Home Camera 1003 performed exceptionally well on the dataset of Danmini Doorbell despite having a lower number of static features. We thus conclude that no significant correlation of the IoT device’s static attribute’s against the performance of transfer learning could be determined. A major reason for this can be due to the fact that the presence of the hardware does not necessarily translate proportionally to its software characteristics; and thereby its network footprint. For example, while almost all devices have a camera module capable of recording videos, this does not reflect on whether a device continuously streams video or only runs periodically or when an event occurs. All of these software behaviors shall result in a different network footprint.

4.2 CIC-IDS2017 Dataset - Anomaly Detection

The CIC-IDS2017 dataset consists of various types of intrusion and DDoS attacks on a variety of endpoints. We trained the anomaly model on a per-device basis using only the benign data points. After the model was trained, we evaluated its capability of detecting the malicious data points.

We observed that for the “Bot” traffic records, the model did not perform sufficiently well. In order to investigate this further, we plotted sample records of the Benign and “Bot” data records in order to visualize the distribution.

The following figures show how some data points belonging to the “Bot” category are very similar to the actual “benign” data. The original paper by the creators of CIC-IDS2017 mentions “Bot” malicious traffic as being generated using “Ares” which is a

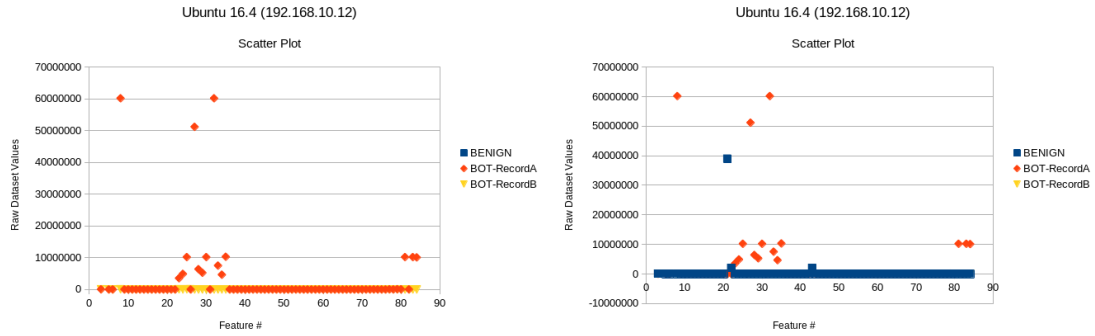


Figure 4.9: A representation of how some malicious traffic does not have significant variance in network footprint

Python-based Botnet that can provide a remote shell, file upload/download, capturing screenshots, and keylogging. Actions such as keylogging and screenshot capturing are unlikely to generate a vastly different network footprint and hence are not distinguishable from a network standpoint.

Figure 4.10 presents the model’s accuracy in successfully detecting the various IDS and DDoS attempts made in the simulated environment. Since DDoS attacks are attempted against only a single endpoint, we excluded the transfer-learning part of the experimentation. Our main aim was to evaluate the validity of autoencoder-based anomaly models against datasets that are more synonymous with the IPFIX [33] format which is an industry-standard. The relatively simple autoencoder-based anomaly model was able to detect DDoS variants with an average accuracy of 90.9%. In addition to the DDoS category, the model performed well on the Web-BruteForce, Web Cross-Site-Scripting, Heartbleed, and Infiltration attack datasets. Accuracy was relatively low against the remaining categories. This points to our earlier discussion on autoencoder lacking the capability of detecting malicious events if clear anomalies are absent in the dataset records.

For the device Windows Server 16, multiple runs of anomaly detection were made on the dataset using a variety of optimizers. While the detection accuracy is mildly varied, the ADAM [35] optimizer and its variants NADAM and ADAMAX had a distinguishably better performance. Among these NADAM was the top performer. Other optimizers that were tested included SGD and ADADELTA. Figure

CHAPTER 4: RESULTS AND DISCUSSION

Device Name	Accuracy													
	Bot	DDoS	DoS (GoldenEye)	DoS (Hulk)	DoS (SlowHttp)	DoS (slowloris)	FTP (Patator)	SSH (Patator)	PortScan	WebAttack-BruteForce	WebAttack-SqlInjection	WebAttack-XSS	Infiltration	HeartBleed
Windows 8.1	61.11%	-	-	-	-	-	-	-	-	-	-	-	-	-
Web Server 16	-	82.58%	97.66%	85.41%	97.80%	91.07%	69.69%	73.41%	96.91%	92.30%	76.19%	97.70%	-	-
Windows Vista	66.78%	-	-	-	-	-	-	-	-	-	-	-	93.75%	-
Ubuntu Server 12	-	-	-	-	-	-	-	-	-	-	-	-	-	100.00%
Windows 7 Pro	60.03%	-	-	-	-	-	-	-	-	-	-	-	-	-
Ubuntu 16.4	75.00%	-	-	-	-	-	-	-	-	-	-	-	-	-
Windows 10 Pro	59.77%	-	-	-	-	-	-	-	-	-	-	-	-	-
Windows 10	50.60%	-	-	-	-	-	-	-	-	-	-	-	-	-

Figure 4.10: CIC-IDS2017 model performance results

Device Name	Metrics	Optimizer	DDoS	DoS (GoldenEye)	DoS (Hulk)	DoS (SlowHttpTest)	DoS (slowloris)	FTP (Patator)	SSH (Patator)	PortScan	WebAttack-BruteForce	WebAttack-SqlInjection	WebAttack-XSS
Web Server 16	ACCURACY	Adam	82.58%	97.66%	85.41%	97.80%	91.07%	69.69%	73.41%	96.91%	92.30%	76.19%	97.70%
	ACCURACY	SGD	80.53%	88.96%	84.33%	91.29%	74.58%	49.98%	49.96%	50.07%	52.02%	52.38%	51.07%
	ACCURACY	NADAM	83.13%	98.28%	84.74%	98.54%	96.67%	49.98%	71.74%	96.45%	92.20%	76.19%	97.70%
	ACCURACY	Adamax	85.76%	97.09%	84.48%	98.21%	94.91%	49.98%	96.38%	96.38%	92.00%	66.67%	97.70%
	ACCURACY	Adadelta	82.61%	85.80%	83.40%	80.65%	73.37%	49.89%	49.96%	50.02%	52.12%	54.76%	51.07%

Figure 4.11: CIC-IDS2017 with different optimization algorithms

CHAPTER 5

Conclusion and Future Work

Our inclination on testing the viability of transfer-learning autoencoder models of IoT devices was based on two rationales.

1. The benign behavior of similar IoT devices on the network should be somewhat similar, and therefore the features learned by the autoencoder model of these IoT devices should be similar too.
2. Since the behavior of DDoS generating malware such as Mirai and Bashlite does not change based on device features, the anomaly introduced by them should be similar too.

Our experimentation positively affirmed that an existing autoencoder neural network can be subjected to transfer learning with limited new data of an unknown IoT device with good accuracy. However, we did not observe a strong relationship between the static features of an IoT device and its normal traffic behavior. We hypothesize that this could be due to the fact that these static features do not adequately represent the functional properties of the IoT device. For example, simply knowing whether an IoT device contains a camera only paints a black and white picture. Whereas network footprint would be impacted by the frequency of the camera's use, its FPS, megapixels, etc.

Our experimentation with the IPFIX formatted data has shown that while *noisy* DDoS traffic may be detected with fair accuracy, this can be improved further. We conclude that building the feature dataset has a significant role in impacting the quality of the learning by the autoencoder. In general, simply focusing on the quantity and size of packets does not provide enough reference points for the neural network to learn a holistic picture.

5.1 Future Work

The following can be interesting future directions;

- The conversion of raw packet captures into feature vectors introduces latency which can undermine the effectiveness of an anomaly detector. Minimizing the role of middlewares converting raw packet-capture (PCAP) files to feature vectors and bringing them into real-time can be explored. The IPFIX framework is widely supported and has the flexibility of configuring custom attributes. This can form an interesting starting point for building a more holistic feature list.

- So far, the anomaly threshold is based on the mean-squared error in the reconstruction Loss and requires a programming logic external to the autoencoder itself. The use of RNN/LSTM can be explored to train anomaly detectors on a time-series input data stream of IoT traffic.

Bibliography

- [1] Yair Meidan et al. *N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders*. Mar. 2018. URL: https://archive.ics.uci.edu/ml/datasets/detection_of_IoT_botnet_attacks_N_BaIoT.
- [2] Ronierison Maciel et al. “Impact of a DDoS attack on computer systems: An approach based on an attack tree model”. In: *2018 Annual IEEE International Systems Conference (SysCon)*. IEEE. 2018, pp. 1–8.
- [3] Zhi-Kai Zhang et al. “IoT security: ongoing challenges and research opportunities”. In: *2014 IEEE 7th international conference on service-oriented computing and applications*. IEEE. 2014, pp. 230–234.
- [4] Guest Author. *Inside the infamous Mirai IoT Botnet: A Retrospective Analysis*. Sept. 2021. URL: [https://blog.cloudflare.com/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis/#:~:text=its%20offensive%20capabilities.-,How%20Mirai%20works,and%20control%20\(C&C\)%20servers..](https://blog.cloudflare.com/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis/#:~:text=its%20offensive%20capabilities.-,How%20Mirai%20works,and%20control%20(C&C)%20servers..)
- [5] Ronald Holt et al. “Deep autoencoder neural networks for detecting lateral movement in computer networks”. In: *Proceedings on the International Conference on Artificial Intelligence (ICAI)*. The Steering Committee of The World Congress in Computer Science, Computer ... 2019, pp. 277–283.
- [6] Ken Dunham and Jim Melnick. *Malicious bots: an inside look into the cyber-criminal underground of the internet*. Auerbach Publications, 2008.
- [7] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. “A detailed analysis of the cicids2017 data set”. In: *International conference on information systems security and privacy*. Springer. 2018, pp. 172–188.

- [8] *A brief history of bots and how they've shaped the internet Today*. Feb. 2022. URL: <https://abusix.com/resources/botnets/a-brief-history-of-bots-and-how-theyve-shaped-the-internet-today/>.
- [9] David Dagon et al. "A taxonomy of botnet structures". In: *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. IEEE. 2007, pp. 325–339.
- [10] Nabil Hachem et al. "Botnets: lifecycle and taxonomy". In: *2011 Conference on Network and Information Systems Security*. IEEE. 2011, pp. 1–8.
- [11] Sheharbano Khattak et al. "A Taxonomy of Botnet Behavior, Detection, and Defense". In: *IEEE Communications Surveys Tutorials* 16.2 (2014), pp. 898–924. DOI: [10.1109/surv.2013.091213.00134](https://doi.org/10.1109/surv.2013.091213.00134).
- [12] Yu Fu et al. "Stealthy domain generation algorithms". In: *IEEE Transactions on Information Forensics and Security* 12.6 (2017), pp. 1430–1443.
- [13] B. Smith. "A Storm (Worm) Is Brewing". In: *Computer* 41.02 (Feb. 2008), pp. 20–22. ISSN: 1558-0814. DOI: [10.1109/MC.2008.38](https://doi.org/10.1109/MC.2008.38).
- [14] CO Emmanuel et al. "On the internal workings of botnets: A review". In: *International Journal of Computer Applications* 138.4 (2020), pp. 39–43.
- [15] Yonglin Zhou et al. "DGA-Based Botnet Detection Using DNS Traffic." In: *J. Internet Serv. Inf. Secur.* 3.3/4 (2013), pp. 116–123.
- [16] Ben Stock et al. "Walowdac-analysis of a peer-to-peer botnet". In: *2009 European Conference on Computer Network Defense*. IEEE. 2009, pp. 13–20.
- [17] Abdul Hannan, Christian Gruhl, and Bernhard Sick. "Anomaly based Resilient Network Intrusion Detection using Inferential Autoencoders". In: *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*. 2021, pp. 1–7. DOI: [10.1109/CSR51186.2021.9527980](https://doi.org/10.1109/CSR51186.2021.9527980).
- [18] Kate Highnam et al. "Real-Time Detection of Dictionary DGA Network Traffic Using Deep Learning". In: *SN Computer Science* 2.2 (2021). DOI: [10.1007/s42979-021-00507-w](https://doi.org/10.1007/s42979-021-00507-w).

- [19] Truong Dinh Tu, Cheng Guang, and Liang Yi Xin. “Detecting bot-infected machines based on analyzing the similar periodic DNS queries”. In: *2015 International Conference on Communications, Management and Telecommunications (ComManTel)*. IEEE. 2015, pp. 35–40.
- [20] Rohan Doshi, Noah Apthorpe, and Nick Feamster. “Machine Learning DDoS Detection for Consumer Internet of Things Devices”. In: *2018 IEEE Security and Privacy Workshops (SPW)* (2018). DOI: [10.1109/spw.2018.00013](https://doi.org/10.1109/spw.2018.00013).
- [21] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.
- [22] Phanindra Reddy Kannari, Noorullah C. Shariff, and Rajkumar L. Biradar. “Network intrusion detection using sparse autoencoder with Swish-Prelu Activation Model”. In: *Journal of Ambient Intelligence and Humanized Computing* (2021). DOI: [10.1007/s12652-021-03077-0](https://doi.org/10.1007/s12652-021-03077-0).
- [23] Andrew Ng et al. “Sparse autoencoder”. In: *CS294A Lecture notes* 72.2011 (2011), pp. 1–19.
- [24] Xianxu Hou et al. “Deep feature consistent variational autoencoder”. In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2017, pp. 1133–1141.
- [25] Dor Bank, Noam Koenigstein, and Raja Giryes. “Autoencoders”. In: *arXiv preprint arXiv:2003.05991* (2020).
- [26] Ren-Hung Hwang, Min-Chun Peng, and Chien-Wei Huang. “Detecting IoT malicious traffic based on autoencoder and convolutional neural network”. In: *2019 IEEE Globecom Workshops (GC Wkshps)*. IEEE. 2019, pp. 1–6.
- [27] Ren-Hung Hwang et al. “An unsupervised deep learning model for early network traffic anomaly detection”. In: *IEEE Access* 8 (2020), pp. 30387–30399.
- [28] Mohammad A Salahuddin et al. “Time-based anomaly detection using autoencoder”. In: *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE. 2020, pp. 1–9.

BIBLIOGRAPHY

- [29] Iman Sharafaldin et al. “Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy”. In: *2019 International Carnahan Conference on Security Technology (ICCST)*. IEEE. 2019, pp. 1–8.
- [30] Zhaomin Chen et al. “Autoencoder-based network anomaly detection”. In: *2018 Wireless Telecommunications Symposium (WTS)*. IEEE. 2018, pp. 1–5.
- [31] Jinwon An and Sungzoon Cho. “Variational autoencoder based anomaly detection using reconstruction probability”. In: *Special Lecture on IE 2.1 (2015)*, pp. 1–18.
- [32] Iman Sharafaldin., Arash Habibi Lashkari., and Ali A. Ghorbani. “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization”. In: *Proceedings of the 4th International Conference on Information Systems Security and Privacy - ICISSP, INSTICC*. SciTePress, 2018, pp. 108–116. ISBN: 978-989-758-282-0. DOI: [10.5220/0006639801080116](https://doi.org/10.5220/0006639801080116).
- [33] B. Claise, B. Trammell, and P. Aitken. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. STD 77. <http://www.rfc-editor.org/rfc/rfc7011.txt>. RFC Editor, Sept. 2013. URL: <http://www.rfc-editor.org/rfc/rfc7011.txt>.
- [34] Unsub Shafiq and Transfer Learning AutoEncoder Neural Networks for Anomaly Detection of DDoS Generating IoT Devices. *Transfer Learning AutoEncoder Neural Networks for Anomaly Detection of DDoS Generating IoT Devices*. Version 1.0. Apr. 2022. URL: <https://github.com/unsubshafiq/autoencoder-transferlearning.git>.
- [35] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).