Low-Rate Attack Detection with Intelligent Fine-Grained Network Analysis

A thesis submitted in partial fulfilment

of the requirement for the degree of Doctor of Philosophy

Baskoro Adi Pratomo

October 2020

Cardiff University
School of Computer Science & Informatics

Declaration

This work has not previously been accepted in substance for any degree and is not concurrently submitted in candidature for any degree.

Signed ..... (candidate)

Date .....

Statement 1

This thesis is being submitted in partial fulfillment of the requirements for the degree of PhD.

Signed ..... (candidate)

Date .....

Statement 2

This thesis is the result of my own independent work/investigation, except where otherwise stated. Other sources are acknowledged by explicit references.

Signed ..... (candidate)

Date .....

Statement 3

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ..... (candidate)

Date .....

A copy of this document in various transparent and opaque machine-readable formats and related software is available at `https://baskoroadi.web.id`.

To my family
for their patience and support.

Abstract

Low-rate attacks are a type of attacks that silently infiltrate the victim network, control computers, and steal sensitive data. As the effect of this attack type is devastating, it is essential to be able to detect such attacks. A detection system allows system administrators to react accordingly. More importantly, when the detection system is to analyse the network traffic, it may identify the malicious activity before the attack reaches the system. And by incorporating machine learning into the detection approach, the Network-based Intrusion Detection System (NIDS) will be able to adapt to evolving attacks and minimise human intervention, unlike signature-based NIDS.

Several works have tried to address the problem of low-rate attack detection. However, there are several issues with these previous works. Some of them are dated; therefore their performance drops on contemporary low-rate attacks. Some of them only focus on detecting attacks in one protocol, while low-rate attacks exist on various protocols. To tackle this problem, we proposed two Deep Learning (DL) models which analyse network payload and were trained with the unsupervised approach. Our best performing model surpasses the state-of-the-arts and provides an improvement in detection rate of at least 12.04%. The experiments also show that payload-based NIDSs are superior to header-based ones for identifying low-rate attacks.

A common approach in payload-based NIDSs is to read the full-length application layer messages, while in some protocols such as HTTP or SMTP, it is usual to have lengthy messages. Processing the full-length of such messages would be time-consuming.

The damage from the attack may have been done by the time the decision for the particular message comes out. Therefore, we proposed an approach that can early predict the occurrence of low-rate attacks from as little information as possible. Based on our experiments, the proposed method can detect 97.57% of attacks by merely reading, on average, 35.21% of the application layer messages. It improves the detection speed by three-fold.

Acknowledgements

This thesis has been a massive challenge for me. I would not have finished it without helps and supports from many people I met during my study.

Firstly, I would like to thank my first supervisor, Pete Burnap. Without his guidance, I would have gotten distracted and lost. He did not give up on me when I almost had given up on myself. He also showed huge patience when dealing with my writing, which helped me to get back on my track.

I would like to thank my second supervisor, George Theodorakopoulos, for the discussions we had during my study. His alternate point of view on various matters had helped to figure out solutions when I had been stuck with some issues.

I would like to thank Lembaga Pengelola Dana Pendidikan (LPDP), Indonesia, for providing funding for this research.

I would like to thank my family for the patience and support. In particular, my wife, Sari, for the constant reminder that brain needs glucose to function, especially during the lockdown. My mum and sister who always pay attention to my health and my well-being, and sometimes, my hair. I dedicate this thesis for my uncle, who passed away during the writing of this thesis. His travels around the world had inspired my younger self to study abroad.

I would like to thank my friends and colleagues. I'd like to thank Hudan, my partner in crime, who was never tired having - sometimes - inconsequential debates despite the

eight hours time difference and who always reminds me that the best thesis is the one that is finished. I'd like to thank Beryl for organising social events and her suggestions on our shared interests which have kept me clear-headed. Stefano, for the borderline banter and life lessons. I will never forget the question you asked in our first meeting. I would like to thank Roberto for sharing his knowledge on language and cultural references. Lauren, for the language and mathematical advice, and helping me to adapt to the local weather. Covid-19 has stopped us from debating whether the window should be open. Oscar, for spending time to read the thesis and the philosophical discussions. Nyala, for statistical knowledge. Matthew, my brother, for inspirational quotes. Joana and Mafalda, for letting me use some space in their house. Lucie, for her knowledge of places. Pete Sueref, for the constant, literally, pushes. Shakil and Alex, for the regular coffee meeting. I'd like to thank the Cyber Visualisation Demo team, Amir, Irene, Matilda, for the cooperation during this time.

I would like to thank the members of the Indonesian Student Association in Wales. It was a great experience to spend time and to have great discussions with them.

I would like to show appreciation to bands in which their music has helped me to get through this difficult time: Avenged Sevenfold, DragonForce, Helloween, My Chemical Romance, Nightwish, Powerwolf, Sabaton (for the historical knowledge), and SpyAir. Their songs have helped me to stay focused, reduce noise, and keep me awake.

Contents

List of Publications

The work introduced in this thesis is based on the following publications.

- B. A. Pratomo, P. Burnap, and G. Theodorakopoulos. Unsupervised approach for detecting low rate attacks on network traffic with autoencoder. In *2018 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pages 1–8. IEEE, 2018

- B. A. Pratomo, P. Burnap, and G. Theodorakopoulos. Blatta: early exploit detection on network traffic with recurrent neural networks. *Security and Communication Networks*, 2020

List of Figures

List of Tables

List of Acronyms

IDS Intrusion Detection System

NIDS Network-based Intrusion Detection System

HIDS Host-based Intrusion Detection System

NLP Natural Language Processing

ML-NIDS Machine learning-based NIDS

OS Operating Systems

DoS Denial of Services

DDoS Distributed Denial of Services

DARPA99 1999 DARPA Intrusion Detection Evaluation Dataset

KDD99 KDD Cup 1999 Dataset

ISCX12 UNB Intrusion Detection Evaluation Dataset 2012

CICIDS2017 CIC Intrusion Detection Evaluation Dataset 2017

KYOTO06 Kyoto 2006+ Dataset

CIDDS Coburg Intrusion Detection Data Sets

DPI Deep Packet Inspection

VM Virtual Machine

MitM Man in the Middle

SMB Server Message Block

HTTP Hypertext Transfer Protocol

SIP Session Initiation Protocol

IP Internet Protocol

TCP Transmission Control Protocol

UDP User Datagram Protocol

ICS Industrial Control System

SDN Software-Defined Network

IoT Internet of Things

ISP Internet Service Provider

SMTP Simple Mail Transfer Protocol

FTP File Transfer Protocol

ML Machine Learning

SVM Support Vector Machine

GA Genetic Algorithm

LR Logistic Regression

DNN Deep Neural Network

DL Deep Learning

CNN Convolutional Neural Network

ANN Artifical Neural Network

NN Neural Network

LSTM Long Short-Term Memory

GRU Gated Recurrent Unit

RNN Recurrent Neural Network

P2P Peer-To-Peer

MSPCA Multi-Scale Principal Component Analysis

PCDTA Probabilistic Counting Deterministic Timed Automata

HMM Hidden Markov Model

SOM Self Organising Map

PCA Principal Component Analysis

DR Detection Rate

FPR False Positive Rate

Chapter 1

Introduction

1.1 Cyber security: threats and detection methods

Computer networks are under attack on a daily basis, and new vulnerabilities appear frequently. Every time a new vulnerability emerges, the network is at risk of a possible new attack. The impact of an attacker exploiting the vulnerability on a computer network varies depending on the type of attack the adversary uses.

As shown in Figure 1.1, McAfee Labs compiled a list of the top ten attack vectors in 2018-2019 based on the number of reported breaches, including: malware, account hijacking, vulnerability, unauthorised access, targeted attacks, code injection, denial of service, defacement, and theft. Attacks like denial of service aim to take down the service, thus preventing legitimate users from accessing the service. These attacks are usually achieved by sending a massive volume of traffic that overloads the targeted service. GitHub, a popular online source code management website, suffered from this attack in 2018 when an adversary flooded their servers with 1.3 Tbps of traffic with 126.9 million packets of data each second [61].

Other types of attack silently infiltrate the victim network, control computers, and steal sensitive data. We refer to this type of attack type as **low-rate attacks**. For instance, a group called TortoiseShell compromised the network of several IT providers in Saudi Arabia [105]. It is thought that the initial infection vector was through a compromised web server. At least one of the victim's web servers showed indication of a *web*

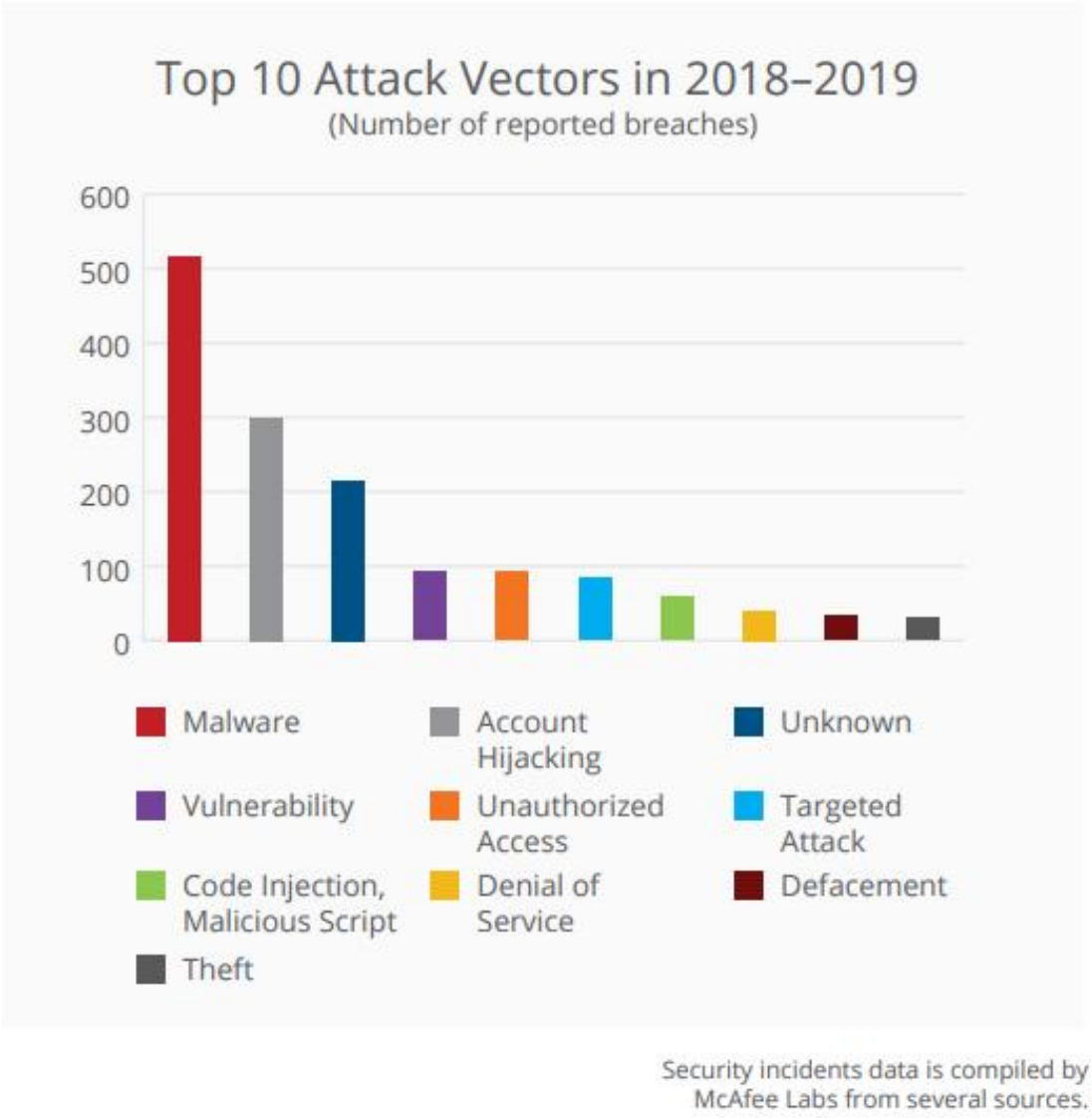

Figure 1.1: Top 10 attack vectors in 2018-2019. Compiled by McAfee Labs. [62]

shell. This is a *web-based backdoor* generally used to upload more malicious files and control the remote computer. The effect of this attack was devastating. In at least two organisations, there was evidence that the adversaries had obtained domain admin-level access. Thus they could easily access any information within the network. The attacking group was still active as of July 2019.

Another incident involving low-rate attacks was an exploit called EternalBlue. It targets a vulnerability in an implementation of file-sharing protocol, Shared Message

Block (SMB), in older versions of Microsoft Windows [8]. The exploit was stolen from NSA and weaponised as ransomware which is known as WannaCry. The WannaCry ransomware that had a global impact, including on the National Health Service (NHS) in the UK [10]. From 12th of May to 18th of May 2017, large numbers of devices in hospitals and surgeries were compromised, causing 19,000 appointments to be cancelled. Although the WannaCry incident in 2018 is arguably noisy, the underlying exploit, EternalBlue, can be used for other purposes and may infiltrate the network silently.

An Intrusion Detection System (IDS) plays an important role in detecting such attacks on computer systems. They monitor networks or systems activities and raise an alert when suspected malicious activity has been identified. Based on its placement and data sources, there are two kinds of IDS: *host* and *network-based* IDS. A Host-based Intrusion Detection System (HIDS) works by examining log files, system calls, or memory content. A Network-based Intrusion Detection System (NIDS) is placed in the middle of internal and external networks, usually near the router, thus it can monitor every incoming and outgoing packet.

One benefit of detecting attacks at the network level is that it enables us to identify attacks *before* they reach the victim (endpoint devices). There are two approaches to achieving this. A *signature-based* Network-based Intrusion Detection System (NIDS) looks for a particular signature/pattern of known attacks or a sequence of bytes that always appears in a particular attack, for example, a *No-Operation* byte (`0x90`) in a buffer overflow attack. The signature/rule database must be updated regularly to cope with new attacks that have never been seen before (*zero-day attacks*). However, network attacks are constantly evolving. Every time the defence side comes up with a signature for the attack, adversaries will often find a way to avoid detection. Encoding attack messages, creating unique attack patterns with polymorphic techniques, or simply replacing known malicious byte values are some techniques commonly used to avoid signature-based detection. To cope with diverse attack patterns, the second type

of NIDS, *anomaly-based* NIDSs are gaining traction as a possible solution. Rather than relying on a database of known malicious patterns, an anomaly-based NIDS is trained to recognise malicious traffic or look for a deviation from usual/legitimate traffic. This way, anomaly-based NIDSs have the potential to detect attacks even when the adversary slightly modify their messages.

An anomaly-based NIDS needs a set of samples to gain insight into the characteristics of network traffic to be able to detect malicious or unusual data flows. A flow consists of multiple network packets being transmitted from one host to another, and in general each packet comprises two parts, headers and a payload as illustrated in Figure 1.2. The usual approach to analyse traffic is to study packet header information, such as IP addresses, ports, packet size, or time-based statistics (i.e., inter-packet arrival time, round trip time, and many others). The information is generally helpful to detect high-rate attacks since they generate a massive amount of packets in a short amount of time, something that does not often happen with legitimate traffic. Low-rate attacks are more surreptitious than high-rate ones. They blend in with legitimate traffic. Thus, it questionable whether packet header data can distinguish malicious traffic from legitimate data flows.

Many low-rate remote attacks work by injecting specially crafted code inside the payload of network packets. Adversaries need the code to be present on the target system to gain or maintain control of the target. The code will likely contain a sequence that would be unlikely to appear in legitimate traffic [39]. Some parts of the transmitted application layer message are consistent in text-based protocols (e.g., HTTP, FTP, SMTP), such as the name of a requested file, HTTP header values, SMTP/FTP commands, and many others. Analysing the payload, instead of the packet header, offers a different perspective on the flow of data within a network and therefore potentially new insights into the malicious traffic destined for a targeted system.

Several anomaly-based NIDSs utilise *machine learning* by building a model of legitimate traffic, looking for any deviations, and marking those deviations as mali-

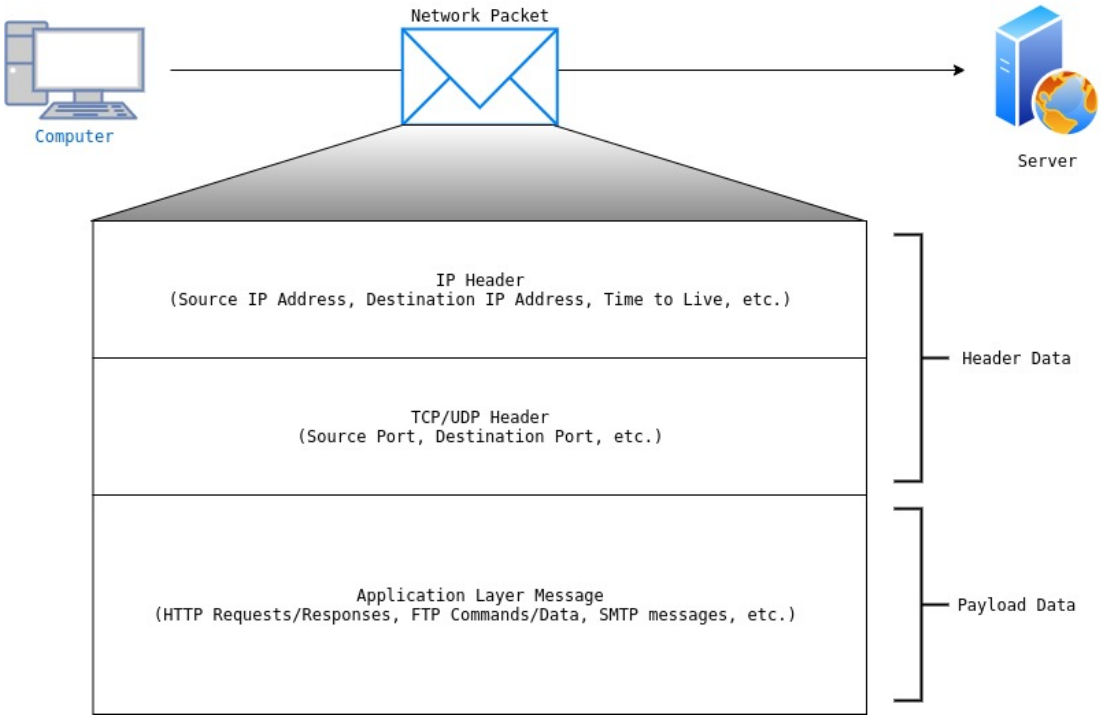

Figure 1.2: An illustration of what is being transmitted between two parties communicating via the Internet. A flow transmits multiple packets from one host to the others. Each packet consists of two parts, headers and a payload..

cious [108, 58, 103]. This approach is referred to as *unsupervised learning*. Others, which are referred to as *supervised learning*, work by training a model with legitimate and malicious samples [19, 115, 51]. These anomaly-based NIDSs with a supervised learning approach can learn the difference between them. Both unsupervised and supervised approaches need representative examples of malicious and legitimate traffic to evaluate their method. However, 18 out of 34 existing works in this area were evaluated with the DARPA99 dataset [100], which was released in 1999 and so is missing representative data for contemporary attacks. It is also known to have other issues which present concerns for the validity of these outcomes [74]. It may lead to a problem as the method might perform well in the experiment setup but poorly in a real-world environment.

This issue was highlighted with experiments conducted by Hadžiosmanović et al. [50]

and Wressnegger et al. [112]. They evaluated several works [107, 24, 108, 86] with a more recent dataset and found that the performance dropped. It shows that as both legitimate and malicious traffic has evolved, the previous approaches drop in ability to detect new and unseen attacks. Since then, newer datasets with more recent malicious and legitimate traffic have been proposed. ISCX12 [98] and UNSW-NB15 [80] are some examples of such datasets. Both are generated in a simulated environment with tools (e.g., IXIA PerfectStorm) to generate malicious and legitimate traffic. However, it remains a question whether those datasets capture representative examples of contemporary low-rate attacks and do not have the similar problem to earlier datasets, i.e., DARPA99.

Apart from constantly evolving, low-rate attacks target various services, each uses different protocols. Table 1.1 and 1.2 show the distribution of exploits in ExploitDB, a website which collects exploits. It is shown that web-based attacks are the most popular with more than 2000 exploits target HTTP services. However, there are still exploits in other protocols, such as FTP, IMAP, SMTP, and SSH.

Similar to the header-based NIDS, a payload-based NIDS also requires a set of features to identify malicious traffic. A common approach of obtaining features from the payload is by parsing protocol messages and taking their attribute values [19, 115, 25]. This feature set captures the representation of a specific protocol better, thus making it easier to identify deviation in the network traffic since this approach would let us know which part of the message indicates that the traffic is malicious. However, each protocol has a particular message format. Therefore, this feature extraction method would require us to create a different set of features for every protocol out there.

Another approach is to use a set of features which are applicable to any protocols. Each protocol may have a particular model, but all models may use generic features, such as frequency of bytes, words, or n-grams [107, 86, 103]. These features are widely used in text classification problem since they can capture behaviour or context of a text document. Since classifying network payload is relatively close to a text

classification problem, it is possible to use the approach for extracting features from network payload. In doing so, we would not have to craft a feature set to represent traffic for each protocol.

While some research in payload-based NIDS used protocol-agnostic features [86, 103, 111], they evaluated their approach on HTTP only. It is not yet known whether their approach can identify low-rate attacks on other protocols. Detecting attacks on various protocols means we will be able to capture more attacks and reduce the possibility of the system being compromised.

Table 1.1: The distribution of exploits in ExploitDB based on the targeted platform..

Platform	# of exploits
PHP	20224
Windows	9691
Linux	2953
Multiple	2679
Hardware	1748
ASP	1540
CGI	777
Unix	323
OSX	310
JSP	275

Furthermore, some research in payload-based NIDS performs attack detection on a per-packet basis [108, 103]. This means they treat each packet individually. They do not consider the possibility of the attack being spread across application layer messages in multiple packets. This may cause packets of the same connection to be treated differently. An alert may be raised for one packet in a malicious connection, but not for the other packets. Other research reconstructs the application layer message[21, 22, 19], thus the detection method sees the complete information about the message. The

Table 1.2: The distribution of exploits in ExploitDB based on the targeted port. The targeted port implies which application layer protocol is used to conduct the attack..

Ports	# of exploits
80 (HTTP)	2043
21 (FTP)	158
443 (HTTPS)	107
8080 (HTTP)	107
143 (IMAP)	48
25 (SMTP)	42
8000 (HTTP)	29
22 (SSH)	27
69 (TFTP)	24
Others	39227

result of this approach would be more accurate as it sees the whole flow. However, an application layer message length varies from less than 100 bytes to several kilobytes. If the detection method needed to wait until the end of the message arrives before making a decision, then it is likely that the attack may have already been executed on the victim host before it is detected. Therefore, an open research problem is how we enhance detection methods with early prediction capability, that is: can we detect a low-rate attack with high accuracy while reading fewer packets in the payload? If possible, the network administrator may be able to block the attack early and minimise or even prevent the damage.

1.2 Contributions

Taking into account several questions we have raised with state of the art in NIDS research, we formulated research questions which will be addressed in the following

chapters:

RQ1 How well do existing validation datasets capture representative examples of contemporary low rate attacks?

RQ2 Given recent research has shown that previous work on low rate attack detection across multiple protocols is dated and performance drops on contemporary low rate attacks, how do we improve the performance of low rate attack detection models to deal with evolving cyber attacks that are increasingly causes damage to corporate networks?

RQ3 Can we predict the occurrence of low-rate attacks with fewer data and earlier in the attack, while still retaining a relationship between the sequence of packets, in order to reduce harm on the system under attack?

To arrive at these research questions, we first conducted a literature review of existing research in payload-based NIDS to highlight the current limitations (see Chapter 2). We then studied existing datasets to determine which ones included recent representative examples of low-rate attacks. We found existing datasets to be somewhat lacking in recent representative samples so undertook a rigorous process to develop a bespoke dataset that contains various low-rate attacks (i.e., exploits, backdoors, shellcode) (see Chapter 3). Afterwards, we performed experiments with various methods to detect low-rate attacks and proposed an improvement by using an unsupervised deep learning model (see Chapter 4).

An application layer message can be extremely long, and as such analysing every byte in the payload delays the detection of an attack. This would also delay the reaction and by then the attack may have been complete. Therefore, we developed a novel early prediction method for low-rate attack traffic that does not depend on analysing the full message (see Chapter 5). In summary, below are this thesis' contributions:

Contributions

C1 A comparative study of state of the art supervised and unsupervised anomaly-based methods for detecting low-rate attacks with features obtained from packet header and payload information to investigate the performance of existing methods and features when distinguishing low-rate attacks from legitimate traffic.

C2 A comparative study of network traffic datasets to better understand how representative they are to be used for evaluating NIDS and a dataset of low-rate attack traffic with state-of-the art attacks, various payloads, and encoders. Additionally, the dataset contains information about the location of the malicious traffic within the payload. The location is useful to analyse whether an early prediction method can detect the attack before it completes.

C3 Unsupervised deep learning models to identify low rate attacks in network traffic, putting aside the requirement to provide malicious samples for the training data. The proposed approach offers an improvement in detection rate at least 12.04% from the previous works.

C4 The first early low-rate attack prediction system on network traffic, which predicts malicious instances as they enter the protected network without analysing the whole application layer messages, enabling the administrator to react faster and possibly minimise the damage.

1.3 Limitations

In this thesis, we have several limitations:

- Our proposed methods do not directly take encryption into account. We are aware that encryption is the biggest challenge in payload-based NIDS as it scrambles the data, causing essential information regarding malicious traffic to be hidden. However, unencrypted malicious traffic is still common. Many webshells have been uploaded through Hypertext Transfer Protocol (HTTP), and this attack is

one of the most common attacks in 2019[62]. Moreover, our approach could still be incorporated with application-layer firewalls, such as ModSecurity. As these firewalls are located on the protected hosts, they can read the decrypted application layer messages without breaking any security mechanism, such as being Man-in-the-Middle to decrypt the message. Apart from that, many works on payload-based NIDS published during 2014-2019 also did not deal with encrypted traffic (See Section 2.2.2).

- Due to the nature of the Machine Learning (ML) techniques used, our proposed methods only work with text-based application layer protocols on top of TCP/IP. Binary protocols contain different challenges and deserve a separate study. Again, text-based protocols are still more prevalent[17]. As this thesis limit the scope to protocols on top of TCP/IP, we excludes any work that detects attacks on network protocols that do not use TCP/IP, such as serial-based protocols in Industrial Control System (ICS) environments (e.g., Modbus, BacNet).

1.4 Thesis Structure

The structure of this thesis is as follows:

Background: A comparative study of state-of-the-art anomaly-based NIDS which analyse either packet headers or payloads. We will discuss in more depth the gaps that exist in this research area and the research questions which direct the following chapters in our thesis. The associated chapter creates contribution C1.

How well do existing validation datasets capture representative examples of contemporary low rate attacks?: We start with a review of existing network traffic datasets containing low-rate attacks. After identifying the gaps in the existing datasets, we propose a low-rate attack traffic dataset which addressed the issues we have found in the existing datasets. The associated chapter creates contribution C2.

An unsupervised approach for detecting low-rate attacks on network traffic: We develop a novel anomaly-based NIDSs with an unsupervised approach that performs network payload analysis. The proposed methods are then compared with state-of-the-art methods. The associated chapter creates contribution C3.

Early prediction of low-rate attacks on network traffic with Recurrent Neural Networks: We develop a novel early prediction method for low-rate attacks which reads a small portion of application layer messages. The proposed method can detect low-rate attacks earlier than the state-of-the-art. The associated chapter creates contribution C4.

Conclusions and future works: A reflection of contributions made in this thesis, as well as discussion for possible future work in this area.

Chapter 2

Background

This chapter discusses the advancement of research in intrusion detection by first giving a brief explanation of types of attacks and reviewing the existing research to identify gaps and limitations. The literature review consists of two parts. The first part identifies all works on anomaly-based Network-based Intrusion Detection System (NIDS)s which use machine learning and packet header information as features. The second part discuss anomaly-based NIDSs that read payload data and how they detect low-rate attacks..

2.1 Why study low-Rate Attacks?

Adversaries can launch various attacks on a target system. Each attack type has different traits. Thus detection approaches might work well for some attack types but not others. Previous research usually mentions which type of attack it tries to identify explicitly.

To better understand the utility of previous works at detecting attacks, it is necessary to understand which attack types they have successfully detected. However, different works have their own terms for naming attack types. Therefore, for the purposes of a common language, attack types are clustered into eleven types based on their definition. Similar attack types are grouped. For example, Probe (in the KDD Cup 99 dataset), Reconnaissance (in UNSW-NB15 dataset), and Analysis (in UNSW-NB15 dataset) are

Table 2.1: A list of attack types definitions.

Attack Type	Definition
DoS & DDoS	An attack that prevents legitimate users from accessing the networks, systems, or applications by exhausting resources.
Probe	An activity that gathers information from targeted networks, systems, or applications.
Fuzzers	An attack that tries to break a program or bypass the protection by sending various data. It includes brute-force attacks.
Botnets	Several connected devices which are controlled by the adversaries to do malicious things, usually distributed denial of services.
Exploits	An attack that triggers a vulnerability in a system and usually causes it to be controlled by the adversaries.
Backdoors	Any mechanism that allows unauthorised access to systems or applications
Worms	A computer program that replicate itself onto other hosts and do something malicious in them.

similar. Thus they can be put in the same cluster. The summary of attack types and their brief definition is shown in Table 2.1.

A *Denial of Services (DoS)* attack prevents legitimate users from accessing the provided resources. Ping of Death is an example of this type of attack in which the adversary sends a malformed ICMP packet, bigger than the maximum packet size. This attack causes the target system to crash or reboot, thus denying access from legitimate users. The term DoS and flooding are generally used interchangeably since flooding attacks attempt to make a system fail as well. To amplify the effect, the adversary utilises an army of controlled devices to send junk messages. This attack is known as *Distributed Denial of Services (DDoS)*. The incident on GitHub where adversaries sent 126.9 mil-

lion packets per second is one of the recent examples of this attack [61].

The army of adversary controlled devices used for a DDoS attack is called a botnet [101]. A botnet has many uses, from sending spam emails to taking down a server in minutes. Therefore, botnet traffic is highly correlated with DDoS traffic.

Another type of attack that behaves similarly to DDoS is a *fuzzer*. It is an attack which aims to break an application by giving the target randomly generated input, hoping the application would raise an error. Due to numerous attempts this attack makes in a short period of time, the target may refuse to serve legitimate users as a side effect. *Brute force* attacks attempt to get into the system by sending a combination of usernames and passwords. These are also classified as fuzzers due to their similar behaviour, throwing numerous attempts in the hope of breaking the authentication.

Probe or *reconnaissance* refers to an activity of gathering information from the targeted network, system, or application. It is usually the initial step in penetration testing to give the tester crucial information which will be useful later, such as how the internal network laid out or what are the operating systems and applications installed. The adversary would then know the weakest link to infiltrate subsequently. A Transmission Control Protocol (TCP) stealth scan is the most popular technique for reconnaissance. This technique sends multiple SYN requests to various ports when the adversary receives a SYN-ACK reply. It is implied that the port is open.

We can see a pattern here. The behaviour of sending multiple requests in a short period of time, hoping to achieve something appears in four attack types mentioned. Time-related features (i.e., inter-arrival packet time, round trip time, and many others) would be able to capture the behaviour of these attack types well[39]. Therefore, we classify these types of attacks as **high-rate attacks**.

In contrast to high-rate attacks, we define a **low-rate attack** as any attack which silently infiltrate networks, systems, or applications to compromise them, steal information, or maintain access. This attack-type includes: exploits, backdoors, worms, ICS malicious

commands and data exfiltration.

An *exploit* is a piece of code that triggers unexpected behaviour from an application. The unexpected behaviour may then be used to escalate privilege or gain control of the system. This attack type also includes malicious instructions to a device which cause the device to do something that is not intended. One example is when an adversary exploit a vulnerability in Whatsapp, which caused them to gain control of victim devices by sending a specially crafted MP4 file which contains a reverse shell code [9]. In this case, the reverse shellcode is usually referred to as the *exploit payload*, the piece of code which will be executed once the vulnerability is successfully exploited.

When an adversary successfully infiltrates a system, they would want to maintain the access, even when the main channel they have used during the initial infiltration has been closed. For that purpose, the adversary would plant a *backdoor* in the compromised system. A web shell is an example of a backdoor. It is a web-based application that allows an adversary to upload files, list directories, add users, and maintain access to the compromised system. It also includes an act of planting the backdoor program itself by transmitting the program through a network connection.

Malicious software that can replicate and distribute itself is called a *worm*. Once a worm exist in the network, it will spread itself without any trigger from a human. WannaCry is an excellent example of a worm. It encrypts files in the computer and then exploits a system vulnerability to propagate itself to other computers in the network.

Based on the above definitions of various attack types, we analysed the existing works on NIDS to determine the proportions of the existing literature that have addressed different attack types. Figure 2.1 shows the percentage of attack types being discussed or considered by existing works on anomaly-based NIDS. DoS and DDoS take the biggest portion in the chart. This illustrates that detecting those attacks (high rate) has attracted the most attention. The proportion increases when all high-rate attacks (i.e., DoS & DDoS, Probe, Fuzzers, and Botnets) are combined. It takes more than 50% of the attack types. This result is not surprising given the fact that many network traffic

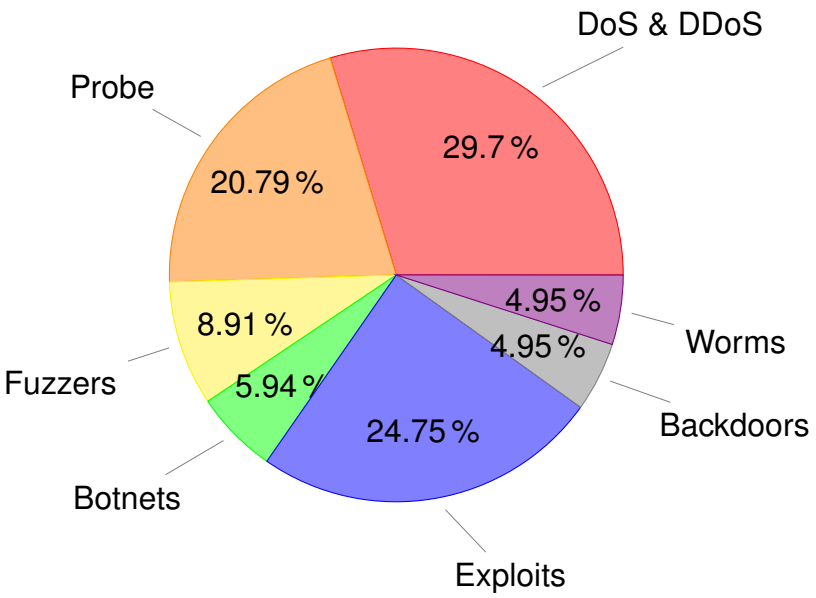

Figure 2.1: The distribution of attack types discussed in anomaly-based NIDS research.

datasets generally provide time-related features and generate others from information found in packet headers (see Section 3.1) which are useful for detecting high-rate attacks.

Where low-rate attacks have been explored, many existing works included them because they existed in the KDD Cup 99 dataset [113, 34, 60, 32, 42, 48, 96] (see Section 3.1). These cases are arguably limited in their findings because the KDD dataset only contains 31.66% of low-rate attack out of 120 [67]. Moreover, as these works did not particularly discuss detecting low-rate attacks. It gives us less information on how their approaches would work on low-rate attacks more generally. This is problematic given that effect of low-rate attacks is severe, costing the victim lots of money, as in the NHS case (£92 million) [10], and ruining their reputation, as in the case of several IT providers in Saudi Arabia [105]. Therefore, this thesis focuses on detecting low-rate attacks.

2.2 Network-based Intrusion Detection Systems

A newly published server on the internet will be targeted by adversaries in minutes or even seconds. It will be scanned for vulnerabilities and exploited as soon as a vulnerability is found as adversaries regularly scan for newly active Internet Protocol (IP) addresses [23]. To protect their network/system, one usually employs a firewall or an Intrusion Detection System (Intrusion Detection System (IDS)). A firewall controls the flow of traffic between networks using differing security postures [99]. It essentially limits access to services by allowing or blocking connections based on predefined rules. For instance, some companies do not allow their internal network to access BitTorrent as it potentially spreads malware, so they drop packets belonging to the protocol. However, firewalls do not prevent attacks to 'allowed' services. For instance, they will not block an SQL injection attack to a company website where web access is allowed.

An IDS is a process or subsystem, implemented in software or hardware, that automates the tasks of (a) monitoring events that occur in a computer network and (b) analyzing them for signs of security problems [99]. Unlike firewalls, IDSs also monitor 'allowed' access, ensuring that it does not contain malicious intent. If a firewall limits access to a network/system, an IDS monitors the limited access that passes through the firewall.

Based on its placement and data sources within the system, there are two kinds of IDS: **host-based** and **network-based IDS**. A host-based IDS works by examining log files, system calls, or memory content. A network-based IDS is placed in the middle of internal and external networks, usually closer to the router facing the internet. It monitors every incoming and outgoing packet. The latter can detect malicious activity before it reaches the protected system. Therefore it may aid the system administrator to react faster and minimise the damage. For that reason, this research will focus more on monitoring network traffic to detect malicious behaviour.

Before further exploring NIDSs, we first clarify the definition of Deep Packet Inspec-

tion (DPI). Broadly speaking, DPI is a data processing technique which inspect the payload of network packets [20]. It is used for many purposes, such as internet censorship, logging data from a particular service, or check for malicious code. Some firewalls have this capability. In that case, DPI is usually used to block traffic to certain websites or limit bandwidth for some services. NIDSs also have this capability if they inspect the content/payload of network packets, but it is for different purpose from DPI firewalls. Although some DPI firewalls also have the capability to detect malicious network traffic, we classify everything that detects malicious traffic as NIDS, including DPI firewalls.

Figure 2.2 shows a general taxonomy of an NIDS. A *signature-based* NIDS has a database of patterns that appear on malicious traffic. For instance, a buffer overflow attack usually contains a byte value `0x90` as a padding for the exploit payload, thus if that value is found on the network traffic, the NIDS will raise an alert. It is highly effective in detecting known attacks given that a good set of rules is provided.

Snort [95] is the most popular open-source NIDS and has been in use since 1998. The signature database is updated regularly, making it the strongest supported signature-based NIDS [5]. BroIDS, now known as Zeek [3], is another powerful network analysis framework that can be utilised as an NIDS. In this area, Zeek is commonly used to extract information from network traffic and parse protocol messages. Another example of signature-based NIDS is Suricata [2]. It was built to overcome Snort's limitation on multithreading hence makes Suricata faster at processing network data. Suricata is also able to read Snort's rules, making it easier to migrate from Snort to Suricata.

The widely known weakness of signature-based NIDSs is that the database needs to be updated regularly. Otherwise, they will not be able to detect *zero-day attacks* (new attacks that have never been seen). Building a sound signature also requires high expertise on the attack. If the signature is too generic, the NIDS will deem legitimate traffic as malicious. However, if it is too specific, it would be easier for the adversary to modify the attack to avoid detection.

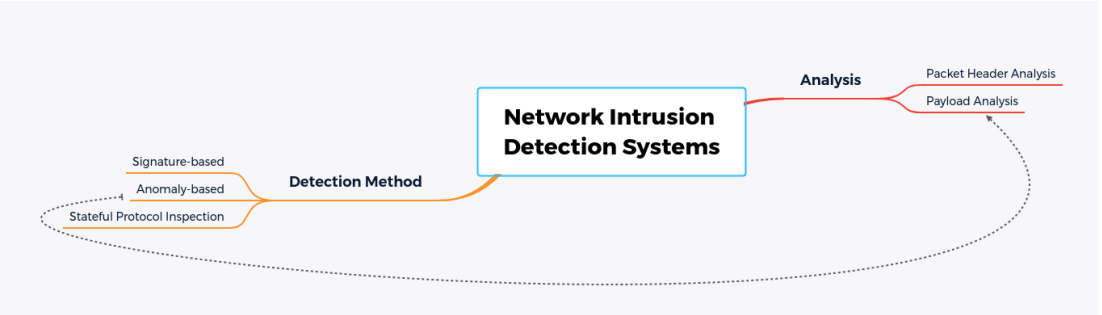

Figure 2.2: A general taxonomy of research on NIDS

To overcome this limitation, the idea of using an *anomaly-based* NIDS was proposed. An anomaly-based NIDS typically learns the behaviour or pattern of legitimate traffic by first building a 'normal' model. It then raises an alert when the new incoming traffic deviates from the 'normal' model. Initially, the approach to build a 'normal' model was by utilising statistical-based anomaly/outlier detection. Then, researchers started using machine learning (ML) algorithms to build a model as they have more predictive power than the statistical-based model [27].

In general, there are two types of learning in Machine Learning (ML), **supervised** and **unsupervised**. In the case of NIDS, a supervised model is trained by learning patterns in samples of all classes, namely legitimate and malicious traffic. **Unsupervised learning**, on the other hand, trains a model by looking at legitimate traffic only, capturing patterns or behaviour within it. Unsupervised learning is similar to statistical-based methods since it is a further development of outlier detection in statistics. Both learning types can still be used to detect malicious traffic. There have been many works employing either supervised or unsupervised learning to identify malicious traffic [39, 100].

An anomaly-based NIDS needs a set of features to be able to distinguish malicious and legitimate traffic. And there are two main approaches of extracting features from raw network traffic. The first approach is to obtain the features from the information presents in the header part of network layer packets (i.e., IP) or transport layer segments (i.e., TCP, User Datagram Protocol (UDP)). IP addresses, ports, TCP flags, packet/segment length are commonly used features [39]. Aggregating values across

multiple packets belonging to the same flow/connection is also a part of this approach [100]. Inter-arrival packet time, the number of packets/bytes transmitted, the average number of retransmitted packets are examples of aggregated values. Since all of the mentioned features are extracted from the header part of an IP packet or TCP/UDP segment, any anomaly-based NIDS which uses them is referred as a *header-based NIDS*.

The second approach goes deeper in the OSI layers by analysing the payload of network packets: the application layer message. An application layer message contains information on what the client wants the server to do, such as: asking for a file, sending a file to be stored, and asking the server to forward an email. This approach analyses the message and decides whether it is malicious by looking at the properties of the message. These properties could be the number of printable characters, frequencies of particular bytes, whether the message satisfies the intended format, or other advanced features [24, 107, 19]. We refer the anomaly-based NIDS that uses this approach as a *payload-based NIDS*.

2.2.1 Header-based NIDS

The advantage of analysing the header of network packets is that it is relatively quick and straightforward to obtain the necessary values as the header only makes up a small portion of the packet size [39]. Additional overhead may occur when aggregating values over several packets, but once the numbers are obtained, they should be fairly quick to process. Moreover, lots of network traffic datasets already provide the preprocessed features which can be fed directly to any ML algorithm. Those datasets help researchers to focus more on the detection algorithm.

Several existing works utilised existing ML algorithms to detect attacks. Xiang et al. [113] proposed the Extreme Learning Machine to detect malicious attempts using Hadoop to process a significant amount of data. Chitrakar and Huang [34] introduced an improvement to incremental Support Vector Machine (SVM) by incorporating unused

support vectors in the next training process. Khammassi and Krichen [60] focused on improving the feature selection process, which in turn should improve the accuracy as well by using a combination of Genetic Algorithm (GA) and Logistic Regression (LR). Wang et al. [110] selected the best features with Information Gain and then experimented with K-Nearest Neighbour (KNN) and One-class SVM. Selvakumar and Muneeswaran [96] aimed toward a similar goal of selecting the best features with Mutual Information Firefly algorithm. Chiba et al. [32] conducted experiments on detecting intrusion with Neural Network and various sets of parameters and improved their work on intrusion detection by introducing an ensemble algorithm of Genetic Algorithm, Simulated Annealing, and Neural Network [33].

Other research proposed new or improved algorithms and evaluated them with an existing dataset. GM Median Nearest Neighbours LDA [42] and an ensemble of SVM with feature augmentation [48] are examples of improvement work on the detection algorithm.

All the works above were evaluated or trained using the KDD Cup 99 dataset [37]. They either proposed a novel detection algorithm or feature selection strategy. None of them introduced a new feature, and the dataset itself leans toward DoS and Probe attacks. Moreover, there is little discussion on what kind of attacks they are trying to detect as different attack types may exhibit unique behaviour. The fact that the dataset is outdated has added a problem to the area. It is hard to judge the relevance of the result to the current situation as attacks have evolved.

There are a few other works on header-based analysis which did not evaluate their method with the KDD Cup 99 dataset. One noticeable characteristic of these works is that they always explicitly mentioned what kind of malicious activities they are trying to detect or what type of network environment they will analyse. Some works on detecting botnets [116, 93, 36], DDoS [29, 87, 30, 65, 16, 91], or both [102, 81, 28]. Some of these works mentioned the type of network environment that they were analysing. Rahbarinia et al. [93] limit the scope of the network to large Internet Service

Provider (ISP) networks. Lee et al. [65] and Chen and Yu [29] proposed a method to detect intrusion in Software-Defined Network (SDN).

Zhang et al. [116] aimed to detect botnets in Peer-To-Peer (P2P) networks (i.e., BitTorrent, Emule, Limewire, Skype, and Ares) by clustering the behaviour of different P2P applications. If a network traffic pattern does not belong to any of the clusters, it will be marked as a potential botnet activity. Rahbarinia et al. [93] proposed an efficient tracking of botnets in large ISP networks by analysing DNS requests transmitted by machines. From those requests, they proposed three groups of features, namely machine behaviour, domain activity, and IP abuse. They then built several classifiers to inspect how their proposed features affected botnet detection. Chen and Yu [29] proposed DDoS detection in SDN environment. They extracted several features from the network header, used a Neural Network (NN) classifier, and implemented it in an SDN environment with OpenFlow [75]. Similarly, Athena [65] also developed a framework for anomaly detection in SDN. It collected SDN control messages and extracted features from them. Chen et al. [30] proposed an improvement to Multi-Scale Principal Component Analysis (MSPCA) to detect DoS attacks better. The MSPCA calculates the Square Prediction Error to see how far the new traffic is from normal behaviour. If the error surpasses a particular threshold, that traffic is deemed malicious. Stergiopoulos et al. [102] employed side-channel features on TCP packets to detect various attacks and experimented with various ML algorithms. Muller et al. [81] proposed an incremental clustering which can prevent training attack, a type of attack which direct a model to consider malicious traffic to be benign. Cid-Fuentes et al. [36] applied One-Class SVM, a widely known novelty detection algorithm, to detect botnets in the ISCX 12 dataset (see Section 3.1). Unlike most other approaches, the model in this approach is continuously updated. Carrasco and Sicilia [28] modelled network traffic with skip-gram [49] and used a neural network to detect anomalies. Despite having 49 features from the dataset (UNSW-NB15, see Section 3.1), the authors only opted IP addresses, ports, and protocol as the feature set which arguably not enough to capture the behaviour of malicious traffic. Mirsky et al. [78] proposed an unsupervised detec-

tion method of DoS based on Autoencoders, a deep learning architecture that may be utilised for outlier detection. Their work was evaluated in an Internet of Things (IoT) environment. Ahmed et al. [16] generated application fingerprints from their network traffic and utilised those to build an unsupervised model to detect DDoS attacks. Qin et al. [91] profile anomalous behaviour with symmetry degree, which is the maximum ratio between the number of internal IP addresses sending packets to external to the number of external IP addresses sending packets to internal networks.

Table 2.2 summarises the aforementioned works. It highlights that the majority of existing work have not taken low-rate attacks into account. Most of them focused on detecting DoS and DDoS (high rate attacks). Carrasco and Sicilia [28] and Stergiopoulos et al. [102] did pay attention to exploits as an example for low-rate attacks[1]. However, [28] work relies solely on IP addresses, ports, and protocols to identify malicious traffic, which makes the result questionable as the malicious traffic in the dataset had been generated by a separate set of IP addresses. Considering that earlier works on header-based NIDS rarely focused on detecting low-rate attacks, it is important to analyse how well header-based analysis can identify low-rate attacks.

[1]Publications by Stergiopoulos et al. [102], Mirsky et al. [78], and Carrasco and Sicilia [28] were published after our paper [88]

Table 2.2: A list of anomaly-based NIDS research which obtained a set of features from information in the header of network packets. Attack types are written based on the definition in Table 2.1. The second column shows whether the work took low-rate attacks into account in their research. A checkmark (✓) shows that the respective work included low-rate attacks in the evaluation.

Authors/Work	**Attack Types Detected (Contain Low-Rate Attacks)**	**Features**	**Detection Method(s)**
Xiang et al. [113]	Probe, DoS, Exploits (✓[2])	KDD Cup 99 features	Extreme learning machine
Chitrakar and Huang [34]	Probe, DoS, Exploits (✓[2])	KDD Cup 99 features	Incremental SVM
Khammassi and Krichen [60]	Probe, DoS, Exploits (✓[2])	KDD Cup 99 features	Decision Tree with GA-LR for feature selection
Chiba et al. [32]	Probe, DoS, Exploits (✓[2])	KDD Cup 99 features	Neural Network
Wang et al. [110]	Probe, DoS, Exploits (✓[2])	KDD Cup 99 features	KNN, One-class SVM
Elkhadir and Mohammed [42]	Probe, DoS, Exploits (✓[2])	KDD Cup 99 features	GM Median NN-LDA
Gu et al. [48]	Probe, DoS, Exploits (✓[2])	KDD Cup 99 features	Aggregated SVM

[2]The work did not specifically discuss low-rate attacks detection as the dataset only contains a small portion of low-rate attacks.

Work	**Attack Types Detected (Contain Low-Rate Attacks)**	**Features**	**Detection Method(s)**
Selvakumar and Muneeswaran [96]	Probe, DoS, Exploits (✓[2])	KDD Cup 99 features	Bayesian Network, C4.5
Chiba et al. [33]	Probe, DoS, Fuzzers, Exploits (✓[2])	KDD Cup 99 features	Neural Network
Zhang et al. [116]	Botnets (✗)	numbers of packets and bytes sent	K-Means + Hierarchical clustering
Rahbarinia et al. [93]	Botnets (✗)	machine behaviour, domain activity, and IP abuse	J48, Random Forest
Chen and Yu [29]	DDoS, Worms (✗)	packet rate, percentage of transport layer packets, IP addresses, ports, inter-arrival packet time, duration, number of packets and bytes, and special protocol of application layer	NN

Work	**Attack Types Detected (Contain Low-Rate Attacks)**	**Features**	**Detection Method(s)**
Chen et al. [30]	DoS (✗)	number of flows, statistical parameters of each flowś packet number (mean, median, and standard deviation), average bytes of flows, average bytes of packets, entropy of IP addresses and ports (source and destination)	MSPCA
Lee et al. [65]	DDoS (✗)	number of packets and bytes, durations, pair flow ratio	K-Means, Gradient Boosted Tree, Decision Tree, Logistic Regression, Naive Bayes, Random Forest, SVM, Gaussian Mixture, Lasso, Linear, Ridge Regressions

Work	**Attack Types Detected (Contain Low-Rate Attacks)**	**Features**	**Detection Method(s)**
Stergiopoulos et al. [102]	Backdoors, Exploits, DDoS, Botnets (✓)	Packet Size, Payload Size, Payload ratio, Ratio to previous packet, Time difference	Logistic Regression, LDA, K-Nearest Neighbour, Decision Tree, Gaussian Naive Bayes, SVM, ANN
Muller et al. [81]	DoS, Botnets, Probe (✗)	number of bytes, packet size, number of concurrent connections, pause since last packet	Stream Clustering
Cid-Fuentes et al. [36]	Botnets (✗)	datasets features	One-class SVM
Carrasco and Sicilia [28]	Fuzzers, Probe, Backdoors, DoS, Exploits, Worms (✓)	header information (srcip, dstip, dport and network protocol (proto))	Skip-Gram with NN
Mirsky et al. [78]	Probe, MITM, DoS (✗)	packet size, packet count, packet jitter	Autoencoders

Work	**Attack Types Detected (Contain Low-Rate Attacks)**	**Features**	**Detection Method(s)**
Ahmed et al. [16]	DDoS (✗)	packet-level and flow-level features	DPMM Clustering
Qin et al. [91]	Probe, Botnet, DDoS (✗)	header information (number of connections)	Symmetry Degree score

2.2.2 Payload-based NIDS

Network payloads/application layer messages contain information about the request or command sent to a remote system. Using this we can determine whether the request or command is intended for malicious purposes. For instance, a remote exploit attack would send a request containing shellcode which comprises a sequence of bytes that do not usually appear in the text-based protocol messages (i.e., HTTP, FTP, and SMTP). The attack may not exhibit distinguishable characteristic on the header-level. In this case, payload analysis would be a better approach at capturing the attack.

Table 2.3 shows the summary of existing works in payload-based NIDS which use either a statistical-based approach (*threshold model*) or a machine learning algorithm. It shows the features the work used, some of which were preprocessed before being fed into the detection algorithm, such as being stored in a particular data structure [94, 103] or turned into a vector matrix [58]. Table 2.3 also gives a brief description of how the research detects the attack. We use the term *threshold model* to refer to a detection method that did not use an ML algorithm, but uses a statistical-based approach to label new incoming traffic as malicious if 'distance' between the traffic and the 'normal' model surpasses the threshold value. Apart from that, datasets used in training and evaluating the model and protocols covered by the research are also mentioned.

As shown in Table 2.3, in constrast to header-based NIDSs, majority of payload-based NIDSs took low-rate attacks into account. Although some works also aim to detect DoS [106, 43, 51] and DDoS [46].

Table 2.3: A summary of works on anomaly-based NIDS which look at network payloads. The list also includes any approach using a threshold model, a statistical model where a threshold value is used to distinguish data in different classes. The second column shows whether the work took low-rate attacks into account in their research. A checkmark (✓) in the Attack Types Detected column shows that the respective work included low-rate attacks in the evaluation. A checkmark (✓) in the Support Multi Protocols column means the method can detect attack and was evaluated on more than one application layer protocols

[i] D=DARPA99, G=GATECH dataset [86], I=ISCX12 [98], SG=self-generated, U=UNSW-NB15 [80], O=Others, BS=BlattaSploit (see Chapter 3)

Work/Authors	**Attack Types Detected (Contain Low-Rate Attacks)**	**Features**	**Detection Method**	**Dataset(s)[i]**	**Support Multi Protocols**
PAYL [107]	Worms (✓)	Relative frequency count of each 1-gram	A threshold model.	D, SG	✓
Poseidon [24]	Probe, DoS, Exploits (✓)	Relative frequency count of each 1-gram.	A threshold model and Self Organising Map (SOM) for preprocessing	D	✓
Anagram [108]	Worms, Exploits (✓)	n-grams stored in a Bloom Filter	A threshold model	D, SG	✗

* D=DARPA99, G=GATECH dataset [86], I=ISCX12 [98], SG=self-generated, U=UNSW-NB15 [80], O=Others, BS=BlattaSploit (see Chapter 3)

Work/Authors	**Attack Types Detected (Contain Low-Rate Attacks)**	**Features**	**Detection Method**	**Dataset(s)[i]**	**Support Multi Protocols**
Rieck and Laskov [94]	Exploits (✓)	n-grams stored in trie data structure.	A threshold model	D, SG	✓
McPAD [86]	Exploits (✓)	$2v$-grams.	Ensemble of one-class SVM	D, G	✗
HMMPayl [21]	Web-based Exploits (✓)	Byte sequence of the application layer messages	Ensemble of Hidden Markov Model (HMM)s.	D, G, O	✗
RePIDS [58]	Exploits (✓)	Mahalanobis Distance Map which is originated from relative frequency count of each 1-gram, filtered by Principal Component Analysis (PCA).	A threshold model	D, G	✗

[*] D=DARPA99, G=GATECH dataset [86], I=ISCX12 [98], SG=self-generated, U=UNSW-NB15 [80], O=Others, BS=BlattaSploit (see Chapter 3)

Work/Authors	**Attack Types Detected (Contain Low-Rate Attacks)**	**Features**	**Detection Method**	**Dataset(s)[i]**	**Support Multi Protocols**
Oza et al. [82]	Exploits (✓)	Relative frequency count of each 1-gram.	Several threshold models	D, G, SG	✗
Whalen et al. [111]	Exploits (✓)	n-grams	Aggregated Bloom Filter, Logistic Regression, Random Forest	I, SG	✗
Bartos et al. [22]	Botnets (✗)	information from HTTP request headers and the lengths	SVM	SG	✗
Anderson and McGrew [19]	Exploits, Backdoors (✓)	Packet header information and TLS, DNS, and HTTP messages	Logistic Regression	SG	✗

[i] D=DARPA99, G=GATECH dataset [86], I=ISCX12 [98], SG=self-generated, U=UNSW-NB15 [80], O=Others, BS=BlattaSploit (see Chapter 3)

Work/Authors	**Attack Types Detected (Contain Low-Rate Attacks)**	**Features**	**Detection Method**	**Dataset(s)[i]**	**Support Multi Protocols**
Golait and Hubballi [46]	DDoS (✗)	Session Initiation Protocol (SIP) messages	Probabilistic Counting Deterministic Timed Automata (PCDTA)	SG	✗
Zhang et al. [115]	Exploits (✓)	Packet header information and HTTP and DNS messages	Naive Bayes, Bayesian Network, SVM	D, SG	✓
OCPAD [103]	Exploits (✓)	n-grams	A threshold model	G, SG	✗
Bortolameotti et al. [25]	Backdoors (✓)	HTTP messages	Clustering	SG	✗
Hamed et al. [51]	DoS, Fuzzers, Exploits (✓)	n-grams of base64-encoded payload	SVM	I	✓
Wang et al. [109]	Backdoors (✓)	n-grams of words	Linear SVM	SG	✗

* D=DARPA99, G=GATECH dataset [86], I=ISCX12 [98], SG=self-generated, U=UNSW-NB15 [80], O=Others, BS=BlattaSploit (see Chapter 3)

Work/Authors	**Attack Types Detected (Contain Low-Rate Attacks)**	**Features**	**Detection Method**	**Dataset(s)**[i]	**Support Multi Protocols**
Tripathi and Hubballi [106]	DoS (✗)	HTTP/2 header values	Chi-Square-based outlier detection	SG	✗
AE-OD (see Chapter 4) [88]	Exploits, Backdoors, Worms (✓)	Byte frequencies of L7 payload	Autoencoder-based outlier detection	U, BS	✓
RNN-OD (see Chapter 4)	Exploits, Backdoors, Worms (✓)	Byte sequences of L7 payload	Recurrent Neural Network-based outlier detection	U, BS	✓
Blatta (see Chapter 5) [89]	Exploits, Backdoors, Worms (✓)	n-grams of L7 payload	Recurrent Neural Network-based methods	U, BS	✓

Before continuing discussion with how existing works in payload-based NIDS detect malicious traffic, we first discuss the datasets they used to evaluate their approach as it shows how well the approach would perform and how easy it would be to reproduce the result. Many older research and some newer ones in this area involved the DARPA99 dataset in their evaluation [107, 24, 108, 94, 86, 21, 58, 82, 115], either as a training set, a testing set, or both. Some others used other publicly available dataset which still provide payload information, such as ISCX12 ([111, 51]) and Gatech datasets ([86, 21, 58, 82, 103]. Many newer research opted to generate their evaluation data, but none of them released the dataset publicly. Some of these data are arguably realistic as they have been captured from real network traffic. On the other hand, since the authors do not release the data to public, probably due to privacy concerns, it is hard to reproduce the result, let alone compare the approaches. Therefore, it remains unknown why researchers decided to use the self-generated dataset. Could it be because the existing datasets do not represent actual malicious or legitimate traffic?

Each application which needs data transmission over the network uses an application layer protocol. Some applications developed a proprietary protocol, while others use open protocols (e.g., HTTP, SMTP, FTP). For example, a web browser communicates with web servers with Hypertext Transfer Protocol (HTTP), emails are transmitted across mail servers with Simple Mail Transfer Protocol (SMTP), people sometimes transfer files to a server with File Transfer Protocol (FTP). Each of these protocols has a specific message format. Therefore, an anomaly-based NIDS needs to treat them in a specific manner if it analyses the payload/application layer message.

Analysing messages of those protocols requires a different set of features. Some works proposed their feature set from extracting information in application layer messages. Bortolameotti et al. [25] and Bartos et al. [22] generated their features from HTTP request URI and HTTP headers, i.e., host, constant header fields, size, user-agent, and language. The authors then clustered the legitimate traffic based on those features. Anderson and McGrew [19] collected features by grouping packets from the same

source and collecting observable metadata, TLS, DNS, and unencrypted HTTP-related information. Zhang et al. [115] tracked DNS and HTTP traffic and calculated pairwise features of two events to see their similarity. These features were obtained from the transport and application layer. One of their features is the semantical similarity between two HTTP requests. Tripathi and Hubballi [106] proposed a method to detect slow-rate DoS attacks on HTTP/2 by using information from HTTP/2 headers values as a feature set. A method by Golait and Hubballi [46] aimed to detect flooding attacks on Session Initiation Protocol (SIP) by analysing the sequence of SIP request names.

Features which are derived from a specific protocol capture specific behaviour of legitimate traffic. However, this feature extraction method has a drawback. A different set of features is needed for every application layer protocol that might be used in the network. It would be better to have features that can characterise messages of each protocol without having to craft individual set of features for every protocol. To cope with this problem, some research borrows a feature extraction method from natural language processing problems since a payload can also be seen as a text document. A common approach in that area for modelling document characteristics is to use n-grams. An n-gram is a set of consecutive items (i.e., bytes, words, letters) with the length of n. n-grams have several aliases depending on the value of n, namely unigrams, bigrams, and trigrams for 1, 2, and 3-grams, respectively. The statistical properties of n-grams are then used to build a machine learning model, as it is usually done in natural language processing. Since network payloads have similar traits to text documents, it is possible to use n-gram models to capture the properties of the payload and identify malicious traffic.

Wang and Stolfo [107] proposed PAYL, which extracts 1-grams from all bytes of the payload as a representation of the network traffic. PAYL trains a model over a set of those 1-grams and applies a simplified Mahalanobis distance to measure how far the new incoming traffic is from the model. Poseidon [24] improved PAYL by preclustering the data using a self-organising feature map, resulting in a smaller number

of clusters and increasing the detection speed. RePIDS [57] calculates the distance between bytes of the payload and stores them in a *Mahalanobis Distance Map* (MDM). It then reduces the dimensionality of the MDM and uses the reduced MDM as the normal model. The distance between a new packet and the model is then calculated. Anagram [108] improved the detection accuracy of PAYL by collecting high order n-grams of bytes, instead of 1-gram, from network packets and counting how many of those n-grams had not been seen in the training phase. The numbers of newly seen n-grams provide a measure of how anomalous new traffic is. Rieck and Laskov [94] stored the n-gram representation of bytes in a *trie* data structure and used various distance functions to detect anomalies. McPAD [86] and its predecessor [85] employed multiple Support Vector Machine (SVM) classifiers and modified n-grams of bytes to $2v$-grams, which take the first and last characters of a sequence of bytes with the length of v. HMMPayl [21] is another work based on PAYL which uses Hidden Markov Models to detect anomalies. Wang et al. [109] obtained n-grams of words in HTTP headers. They used special characters, such as ":", ",", space, and "&" to segment the payload into words. Hamed et al. [51] collected n-grams of base64-encoded payload from the traffic as the dataset they were using stores the payload in base64-encoded form.

Taking a look back at Table 2.3, it is shown that some existing works on payload-based NIDS were evaluated with an old dataset, i.e., 1999 DARPA Intrusion Detection Evaluation Dataset (DARPA99). These works then questionably perform good when facing contemporary low-rate attacks. Hadžiosmanović et al. [50] tried to answer that question by conducting experiments with a more recent dataset and PAYL [107], Poseidon [24], Anagram [108], and McPAD [86]. They showed that the performance of those aforementioned works dropped. Another experiment by Wressnegger et al. [112] with Anagram showed the same result.

DARPA99 have issues with not having representative samples of contemporary legitimate and malicious traffic anymore [74]. Thus, researchers have generated network

traffic datasets which arguably include recent attacks, such as ISCX12 [98], UNSW-NB15 [80], and others. Similar to DARPA99, most of them were generated in a simulated environment with a tool like IXIA PerfectStorm and contain both malicious and legitimate traffic.

In order to properly evaluate low-rate attack detection, it is important to have a representative samples of such attacks. Using a more recent tool to generate attacks does not guarantee that the resulting traffic will be representative. Analysing the resulting traffic is still necessary to determine whether existing datasets capture representative and realistic behaviours of contemporary attacks, particularly low-rate attacks. This leads us to our first research question:

RQ1: *How well do existing validation datasets capture representative examples of contemporary low rate attacks?*

More recent works provided improvements from the earlier ones, but they either did not focus on detecting low-rate attacks [116, 93, 29, 87, 30, 65, 81, 36, 78] or covered only one protocol, mostly HTTP [111, 82, 51, 97, 109], despite utilising protocol-agnostic features. Although HTTP is the most popular protocol on the internet, the statistics from ExploitDB in Table 1.2 show that low-rate attacks on other protocols do exist. Being able to detect more attacks would reduce the possibility of the system being compromised. This issue then led us to our second research question:

RQ2: *Given recent research has shown that previous work on low rate attack detection across multiple protocols is dated and performance drops on contemporary low rate attacks, how do we improve the performance of low rate attack detection models to deal with evolving cyber attacks on a range of protocols that are increasingly causing damage to corporate networks?*

In research on payload-based NIDS, there are two common approaches on when to begin detection. Firstly, the method would read the payload of an individual IP packet and started detecting once it received a packet [107, 108, 24, 58, 86, 103, 94]. Existing

approaches to this disregard the relationship between packets, which may lead the detection method to mark packets belonging to the same connections differently. One could be deemed as malicious while others not. Therefore, performance measurement might not be accurate.

Another approach is to reconstruct the TCP streams and extract features from the whole message or use the reconstructed payload information from the dataset [21, 22, 19, 46, 115, 25, 51, 106]. In doing so, the classifier would have the same decision for all packets belonging to the same connection and complete view of the message, hence making a better decision. However, some protocols may have very long messages which need many packets to be sent, such as SMTP, HTTP. However, this approach may be time consuming as the NIDS would have to wait until the end of the message to arrive. As there is no existing model which provides early prediction of low-rate attacks, we argue that this is an open research area needs addressing. By conducting early prediction, we could potentially react quickly and minimising the damage. This issue leads us to a question:

RQ3: *Can we predict the occurrence of low-rate attacks with fewer data and earlier in the attack, while still retaining a relationship between the sequence of packets?*

2.3 Conclusions

Detecting low-rate attacks is an essential task but sometimes overlooked. Nevertheless, the effect of them successfully executing on a system is severe, leading to economic and reputational costs to the victim. This leads us to focus on detection of low-rate attacks in network traffic. Low-rate attack detection has received attention by security researchers - yet there are several issues still to be addressed, such as: the existence of representative examples of contemporary low-rate attacks to validate methods in this area; previous works having performance drops on contemporary low-rate attacks; and no existing works have yet addressed the problem of early prediction of low-rate

attacks. This chapter highlights these issues and gives the following contribution:

C1: *A comparative study of state of the art supervised and unsupervised anomaly-based methods for detecting low-rate attacks with features obtained from packet header and payload information to investigate the performance of existing methods and features when distinguishing low-rate attacks from legitimate traffic.*

Chapter 3

How well do existing validation datasets capture representative examples of contemporary low rate attacks?

Evaluating the performance of an Network-based Intrusion Detection System (NIDS) is usually conducted by running it over data that contain malicious and legitimate traffic. The number of detected malicious traffic and misclassified legitimate instances are then counted. However, to be confident with the result, the data used to evaluate the method must resemble the real-world situation. Moreover, attacks are evolving, so while evaluating an anomaly-based NIDS with an old dataset may yield a good performance - the result may not be representative of its ability to detect contemporary attacks. The performance may drop when it faces more recent attacks as was shown in the experiments conducted by Hadžiosmanović et al. [50] and Wressnegger et al. [112].

DARPA99 is the most widely used dataset in this area. Some issues with this dataset are mentioned by McHugh [74]. Several newer publicly available network traffic datasets have been published since then, such as ISCX12 [98], UNSW-NB15 [80]. Since now the community has more options for evaluating their approach, it is important to

analyse whether those options are appropriate for contemporary low-rate attack detection. This leads to our first research question:

RQ1: *How well do existing validation datasets capture representative examples of contemporary low rate attacks?*

We will first discuss existing datasets that contain low-rate attacks and highlight their issues and limitations. Afterwards, we propose a new low-rate attack traffic dataset to complement the existing datasets.

3.1 Existing Network Traffic Datasets with Low-Rate Attacks

There are many network traffic datasets available, but since we focus on detecting low-rate attacks, we will only discuss such datasets that contain low-rate attacks. Therefore, we also exclude datasets which are not clearly labelled. Some datasets, i.e., KYOTO06, CAIDA, Digital Corpora, and MAWI, do not have a label for each connection in the dataset which would make the evaluation difficult. Therefore, we analysed DARPA99, KDD99, NSL-KDD, ISCX12, UNSW-NB15, and GATECH dataset in the following sections.

We proposed six metrics to measure how well these datasets for evaluating low-rate attack detection methods: how many years ago the traffic was generated, whether the raw payloads can be found in its PCAP files, whether each attack is labelled corresponding to its attack category, the proportion of low-rate attack instances, whether the dataset contains duplicate records, and whether there are low-rate attacks over multiple application layer protocol. Another metric we are going to use is whether the dataset contains legitimate traffic as well as malicious traffic. This metric is not essential to measure how well the datasets are, but it is still worth noting which dataset has samples of legitimate traffic. Before going further, we will first explain why these metrics were

chosen.

Network-based applications keep evolving, and detection methods have to be able to identify attacks in recent traffic. A dataset with obsolete network traffic would not be a good sample for evaluating NIDS. The difference between the real-world network traffic and the one in the dataset would be too far. A detection method may perform well with the dataset but poorly when faces the real-world network traffic. An average exploit has a life expectancy of 5.39 and 8.84 years [14]. Exploits older than that are unlikely to be used anymore as most systems will have been patched against it. Those attacks are unlikely to be seen in the real network. We argue that period when the dataset was generated influence how close it is to the real-world traffic. Therefore, we aimed to use datasets that had been generated no more than five years ago. Five years was selected so that the result of this analysis will still be applicable for the next three years as the upper limit is 8.84 years.

Datasets, such as KDD99 and NSL-KDD, remove the raw payload and do not provide the PCAP files where the data were generated from. As shown in Table 2.3, fifteen works relied on payload information to identify low-rate attacks. KDD99 and NSL-KDD do not provide such information. Thus, researchers would have limited information about the network traffic.

Labels are one of the most essential components in a dataset. Without good labels, it is difficult to measure the proposed method. Datasets commonly have the information on whether a specific record is malicious or legitimate. As we are more focused on low-rate attacks, it is important to know which malicious traffic was generated by a low-rate attack. Therefore, attack type labels are essential. Without it, we would not be able to take specifically low-rate attacks from the dataset.

As most dataset in comparison contains both high and low-rate attacks, it is needed to know the proportion of the low-rate attack instances in the dataset to know whether there are enough samples.

The number of samples does not give the full picture of the dataset. We also need to know whether the dataset contains duplicate records. Duplicate records are an important aspect to consider for Machine Learning (ML)-based NIDS as they may cause the algorithm to bias towards more occurring records. Although duplicate records may be removed during the preprocessing step, it is still preferable if the dataset does not contain them.

Finally, there are multiple protocols on the internet. We use Hypertext Transfer Protocol (HTTP) for web browsing and Simple Mail Transfer Protocol (SMTP) for sending emails. Having representatives for low-rate attacks on multiple protocols would be better to evaluate our method with more varied environments.

1999 DARPA Intrusion Detection Evaluation and KDD Cup 1999 Datasets

In 1999, Defense Advanced Research Projects Agency (DARPA) released a second evaluation dataset for IDS[68]. They simulated real network traffic by generating malicious and benign requests to the network, then captured all network packets by using tcpdump. This is the dataset that then known as 1999 DARPA Intrusion Detection Evaluation Dataset (DARPA99).

DARPA99 consists of three weeks of captured packets for training and two weeks for testing. The first and third week are attack free and usually used to train an Intrusion Detection System (IDS) model. The second, fourth, and fifth weeks contain attacks of different categories, and the last two weeks are for testing. Labels are provided in the form of text files that are too complicated to relate with the packets in the PCAP files. A sample of a label is shown in Fig. 3.1

DARPA99 is still quite popular. As shown in Chapter 2, Table 2.3, this dataset was used nine times as a benchmark. McHugh [74] argued that the dataset was not realistic enough to simulate a real network. Then Mahoney and Chan [72] added weight to this argument by running their IDSs on DARPA99 and real network data, which resulted

Figure 3.1: The label of a connection in DARPA 99 dataset

```
ID: 41.084031
Date: 03/29/1999
Name: ps
Category: u2r
Start_Time: 08:18:35
Duration: 00:46:05
Attacker: 209.154.098.104
Victim: 172.016.112.050
Username: haraldl
Ports:
        At_Attacker: 80{1}, 6000{2}
        At_Victim: 23{3}
```

in very different outcomes. Brugger and Chow [26] also ran Snort[95] over DARPA99 and the result was poor. They suggested that it was caused by either the failure of the dataset to model the attack correctly or Snort did not have suitable signatures for old attacks. Moreover, this dataset only contains 18.97% of low-rate attack instances out of 190 overall attack instances in the dataset [68], which may make the performance of a NIDS evaluated with it biased towards the other types of attack, for instance Distributed Denial of Services (DDoS) and Probe attacks.

In the International Conference on Knowledge Discovery and Data Mining, there was a data mining competition which used a preprocessed version of the DARPA98 to evaluate the submitted approaches. It is known as KDD Cup 1999 Dataset (KDD99) [37]. The committee extracted 41 features from the raw network traffic, in the form of PCAP files, and stored them into a comma-separated file. Each connection was then labelled as normal or attack. Similar to DARPA99 dataset, this dataset considers four attack types, namely denial of service (DOS), unauthorised access from a remote machine (R2L), unauthorised access to local superuser privileges (U2R), and probing. These categories are further divided into specific attack techniques, e.g., buffer overflow, IP sweep, neptune.

Since this dataset was generated from the DARPA98 dataset, an earlier version of DARPA99, it has the same problem of not being realistic enough [74]. This dataset also contains many duplicate records which would bias the model [104].

Despite all the criticism, the use of KDD99 dataset is surprisingly still prevalent in IDS research. In Chapter 2, Table 2.2, it is shown that this dataset was used by eight works, some of them were published in the last five years. The fact that this old dataset is still used to evaluate modern approaches is concerning.

NSL-KDD

Aware of the problem in the KDD99, Tavallaee et al. [104] proposed a few fixes to it. They filtered out redundant records in the training and testing set. Therefore classifiers would not be biased towards more frequent records. They also made the number of records in each group/class proportional. Thus the evaluation result would be more balanced. Finally, they selected a small portion of randomly selected records, enabling everyone able to run an experiment over the same set. Thus, the results will be comparable.

Similar to the KDD99, NSL-KDD is a preprocessed traffic dataset. It provides CSV files with pre-selected features which were directly taken from the KDD99. It may have a different number of records, but the features are exact copies.

This dataset does help to make results across many experiments more comparable. However, this fact does not eliminate the problem that the KDD99 traffic is not a representative of a modern real-world situation. This problem is also raised by Tavallaee et al. [104], yet they argued that NSL-KDD can still be used to evaluate anomaly-based NIDS approaches.

UNB Intrusion Detection Evaluation Dataset 2012 (ISCX12)

Due to the critiques of DARPA 99 dataset, Shiravi et al. [98] generated a network traffic dataset called ISCX12. To make it as close as possible to realistic traffic, they analysed real network traces, built profiles of the traffic, and used those profiles to simulate user behaviour. The traffic generated by the simulation was then captured and used as samples of legitimate behaviour. In doing so, this dataset claimed not to have privacy-related problems, yet be sufficiently realistic.

ISCX12 consists of seven days of network traffic from 11 to 17 June 2012. The only clean, attack-free set is the 11th June - all the others contain attack traffic. Most of them are HTTP-based Denial of Services (DoS), IRC botnet, and SSH brute force attacks. This dataset also has attacks that stem from a PDF buffer overflow vulnerability and SQL injections. Most packets, whether they are benign or malicious, are HTTP packets, though it also contains other protocols (i.e. FTP, SMTP, and IMAP) in smaller samples. It makes the proportion of protocols imbalanced and overall detection accuracy can be misleading. When taking average results of detection, a great result on HTTP might hide poor results on other protocols.

Shiravi et al. [98] released the PCAP files and provide preprocessed information in XML format. Each record in the XML file consists of a tuple of transport layer protocol (i.e., source & destination IP addresses and ports), TCP flags, number of bytes & packets, captured time, direction, and the payload encoded with base64. Connection labels are written in the XML files as well. Unlike the DARPA99 or KDD99, ISCX12 only labels a connection as Normal or Attack, no attack classes present. Although the tag *appName* mentions what application (e.g., HTTPWeb, DNS, SSH) generates the respective traffic, it is difficult to determine the type of attack. Therefore, it is difficult to see if a proposed method would detect low-rate attacks as they are confused with high-rate attacks.

Another issue we found after examining ISCX12 is duplicate records. Several TCP

connections have two XML records. One usually contains the payload while the other does not. Hence, extra care in removing these duplicate records is needed prior to using this dataset. Alternatively, they could use the provided PCAP files instead.

UNSW-NB15 Dataset

Moustafa and Slay [80] generated the UNSW-NB15 dataset by running an IXIA PerfectStorm, a tool to emulate traffic, in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS). The traffic in this dataset was captured in two days, on the 22th January and 17th February 2015. It contains nine attack classes: Fuzzers, Backdoors, Denial of Service, Exploits, Generic, Reconnaissance, Shellcode, and Worms as described in Table 3.1. Apart from PCAP files, UNSW-NB15 also provides the preprocessed data in CSV files and the output of flow information extraction tools (Argus [70] and Bro-IDS, now known as Zeek [3]). Each record in this dataset is labelled with its categories (normal/attack) and its attack type, making it more suitable for research which addresses a specific type of attacks.

The UNSW-NB15 dataset proposed 47 features (excluding the attack category and label). These features are clustered into flow, basic, content, time, and additional related features. Flow features consist of the transport layer tuple, i.e., IP addresses, ports, and protocol type. Basic features include packet-based and flow-based features, such as duration, number of bytes, and number of packets. Content features do not necessarily explore the content/payload of a connection. They contain some TCP-related information, such as window advertisements or sequence numbers. Time features, as the name implies, include various features related to time, for example: jitter, start & end time, and inter-packet arrival time. Lastly, additional generated features are divided into general purposes and connection features.

In general, this dataset is a significant improvement to the DARPA99. However, this dataset also comes with some issues. For payload analysis it is worth noting that some exploits and worms in this dataset are barely distinguishable from legitimate traffic.

For example, several exploit attacks in this dataset send a short HTTP request message containing nothing but a simple HTTP GET request to a non-existent file. Another issue we found is the Host header in several HTTP requests is an arbitrary string. It may affect the detection performance if the research relies on the occurrence of specific bytes as legitimate HTTP requests do not have this behaviour. Therefore, it may be beneficial to complement this dataset with additional attack traffic.

Table 3.1: Attack description in the UNSW-NB15 dataset.

Class Name	Description
Normal	Legitimate traffic without any malicious payload
Analysis	Port scanning, spam, and HTML file injection
Backdoors	Planting malicious files as an alternative entrance to the server
DoS	Denial of service attack
Exploits	Attacks that exploit vulnerabilities in the server to make it doing something unintended
Fuzzers	Sending various inputs to an application to find vulnerabilities
Generic	Attacks that try to break block ciphers
Reconnaissance	Used to gather information of the server
Shellcode	Small piece of code which used to run a program on the remote server
Worms	A piece of code that can replicate themselves, sometimes over the network as well

Gatech Dataset

Perdisci et al. [86] captured seven days of HTTP requests to the College of Computing School website at Georgia Institute of Technology. The traffic in this set is completely unlabelled and deemed legitimate. However, the legitimate traffic data do not seem

to be publicly available, since this would have privacy issues due to containing non-anonymised sensitive data. Perdisci et al. [86] only released the attack traffic dataset they had generated. It is publicly available on his website. Some existing works refer to this dataset as Gatech or McPAD dataset. For the rest of the thesis, we will refer to this dataset as Gatech dataset.

Perdisci et al. [86] ran 66 HTTP-based exploits against a vulnerable system and captured the traffic. Eleven of these attacks contain shellcode and eight of them were used to generate polymorphic attacks. Some attacks have modified payload to avoid detection by an NIDS. As they carefully craft the attack so that the traffic contain real working exploits, it is safe to say that the attacks in this dataset are realistic. Therefore, this dataset has been used to evaluate payload-based NIDSs [82, 103, 86, 85].

The first issue we found in this dataset is the removal of HTTP reponses. Perdisci et al. [86] removed the HTTP responses from the PCAP files so that the dataset only contains unidirectional flows toward the victim machine. Thus, it is unclear what the responses from the victim machines are, so not necessarily representative of real-world traffic flows. Another issue with this dataset is the exploits they used are quite old by today's standards. Most of these exploits were published in 2001-2003 (i.e., MS01-033, MS01-044, MS03-022). Lastly, this dataset only has HTTP-based attacks, no attacks on other protocols exist in the dataset.

Table 3.2: A comparison of publicly available network traffic datasets.

Dataset	Traffic captured less than 5 years ago	PCAP files provided	Attack type label provided	Proportion of low-rate attacks	No duplicate records	Contain legitimate traffic	Contain more than one L7 protocol
DARPA99	✗	✓	✓	36/190 (18.97%)	✓	✓	✓
KDD99	✗	✗	✓	37/120 (31.66%)	✗	✓	✓
NSL-KDD	✗	✗	✓	37/120 (31.66%)	✓	✓	✓
ISCX12	✗	✓	✗	unknown	✓	✓	✓
UNSW-NB15	✓	✓	✓	27K/2,540K (1.06%)	✗	✓	✓
Gatech Dataset	✗	✓	✗	173/173 (100%)	✗	✗	✗
BlattaSploit	✓	✓	✓	5,687/5,687 (100%)	✓	✗	✓

We have discussed strengths and limitations specific to each dataset which are summarised in Table 3.2. To conclude, older datasets, such as DARPA99, KDD99, and NSL-KDD, only have a small proportion of low-rate attacks. ISCX12 was generated with more recent traffic but lacks information on attack types for classification purposes. UNSW-NB15 improved on that matter since it has a clear label of attack types. However, some low-rate attack traffic in UNSW-NB15 contain arbitrary byte sequence and non-working exploits, making these attacks indistinguishable from the legitimate traffic. To cope with this issue, Gatech dataset may be an option to complement UNSW-NB15 as it provides a better representation of low-rate attack traffic. However, it lacks contemporary low-rate attacks.

Malicious low-rate attack traffic may contain background packets as part of the protocol message; thus, the victim will be able to respond accordingly. For instance, an SQL injection attack to a website would not be successful if the attacker only sent a malicious SQL query to the server without sending the required HTTP request along with it. None of the aforementioned datasets have information on which part contains malicious byte sequence and which one is the part of protocol messages. It is difficult to validate whether a detection by a payload-based NIDS is actually triggered by the malicious sequence. It is also not possible to judge how early a payload-based NIDS can predict the occurrence of a low-rate attack because none of the previously used datasets log where the attack actually takes place within the network payload. Therefore, to supplement UNSW-NB15 dataset, we proposed a new low-rate attack traffic dataset, **BlattaSploit**.

3.2 BlattaSploit Dataset

BlattaSploit contains recent low-rate attacks with diverse sets of exploitation techniques, protocols, and NIDS evasion techniques. Each exploitation technique was launched with various *attack payloads*, malicious code carried by low-rate attacks to

be executed in the victim machine, so that it would generate various network payload. To ensure the generated traffic had variation, the attack payload is also encoded with different encoding techniques, e.g., base64 encoding, Shikata Ga Nai, XOR encoding. The encoding technique also makes the malicious payload supposedly harder to detect by a signature-based NIDS.

Metasploit is a penetration testing framework which allows researchers to use various attack vectors to enumerate services, exploit vulnerabilities, or maintain access to the compromised system [12]. We can utilise these various tools in Metasploit to generate attack traffic, by executing them against vulnerable hosts.

Following the typical approach to generating network traffic datasets using a simulated environment [98, 80, 86], we set up two VirtualBox Virtual Machine (VM)s that acted as vulnerable servers to be attacked. The first server was installed with Metasploitable 2 [13], a vulnerable Operating Systems (OS) designed to be infiltrated by Metasploit and contains vulnerable services, e.g., vsftpd, apache httpd. The second VM was installed with Debian 5, vulnerable services, and Wordpress 4.1.18. Both servers were set up in the victim subnet while the attacker machine was placed in a different subnet. These two subnets were connected with a router which was used to capture the traffic, as depicted in Figure 3.2. The network topology is rather simple and far less complex than what we would have in the real world. But since we focus more on the generated application layer messages/payloads, this setup still generated representative data as the payloads are intact regardless of the number of hops the Internet Protocol (IP) packets have to go through.

The attacking VM ran Kali Linux 2016.02 and Metasploit 4.14.27, while the router ran Ubuntu 16.04. The router was set up to block any outgoing traffic to the internet to prevent irrelevant packets from being captured, such as checking-for-update messages sent by the hosts to the repository server. It also prevented the vulnerable VMs from getting attacked from the Internet during the process.

Metasploit has a collection of working attack techniques and is updated regularly. Each

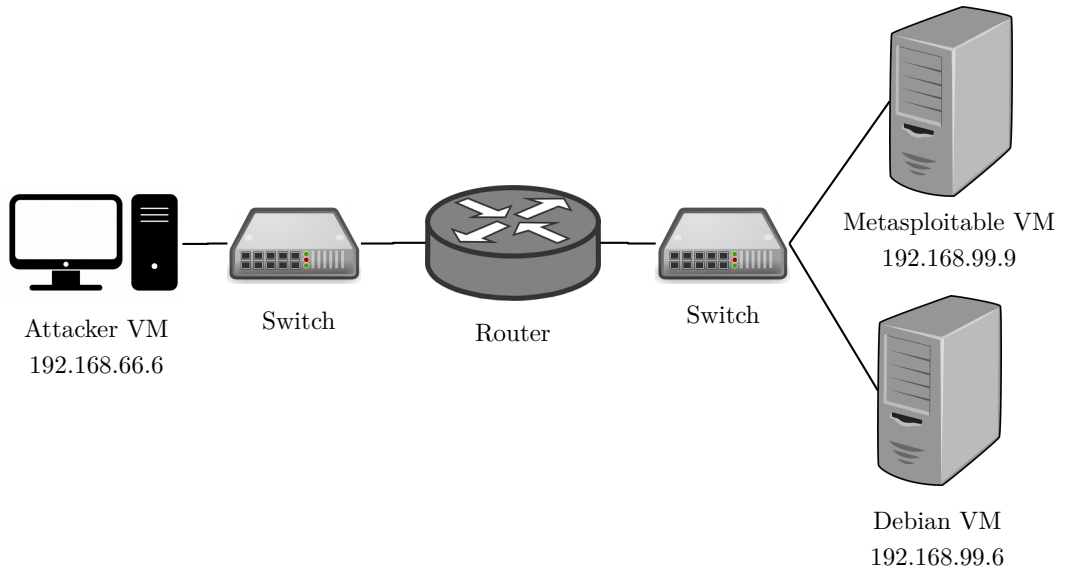

Figure 3.2: The network topology for generating exploit traffic. The Attacker VM running Metasploit and the vulnerable VMs are placed in different network connected by a router. This router is used to capture all traffic from these virtual machines..

attack technique comes with a set of attack payloads and encoders. An **attack payload** is a piece of code which is intended to be executed on the remote machine, whilst an **encoder** is a technique to modify the appearance of particular exploit code to avoid signature-based detection. As mentioned earlier, we executed the attacks against the vulnerable servers with various combinations of attack payloads and encoders to obtain diverse malicious application layer messages.

Each set of attack techniques, a payload, and an encoder were executed at a particular time. No other sets were executed when an exploit was running so that we would know from which set the packets belonged to, and the generated traffic was stored in a separate PCAP file. In other datasets, timestamps are normally used to mark which packets belong to a class, but this information would not be reliable since packets may not arrive in order and if there is more than one source sending traffic at a time, their packet may get mixed up with other sources. Therefore, we decided to separate

network traffic of attacks into separate files.

After the capturing process had finished, we manually analysed the generated PCAP files. We found that not all of the exploits had sent a payload to the victim machine. We then excluded any traffic that: had not sent packets beyond TCP handshaking, had sent only a login request with an arbitrary username and password and failed to get a response from the victim, and had sent a request for non-existent files. This exclusion was conducted to avoid including attacks that can also be considered as high-rate attacks, i.e., fuzzers and bruteforcing. After the filtering, we ended up with 5,687 distinct TCP connections. Finally, we preprocessed the PCAP files with tcpflow [44] to obtain the application layer message for each TCP connection.

A summary of this dataset is shown in Table 3.3, while the complete list of attack techniques used to generate this dataset is shown in Table A.1. 81.25% of attacks in this dataset were disclosed after 2010 and 29 of them were published after 2015, which means the majority of attacks in this dataset are fairly recent and targeting various applications. Although HTTP takes a considerable amount of traffic in this dataset, the BlattaSploit dataset still contains attacks on other protocols, namely FTP, SMTP, and POP3. This dataset also has various type of attack payloads, e.g., JavaScript, PHP, Perl, Python, Ruby, Shell script, SQL, and byte code/opcode for shellcode-based exploits. We obtained the type of attack payload by manually observing the generated traffic. We also marked the position of the attack payload in the application layer message so that we know where the malicious part of the message really is. This information will be useful to measure how the performance of early attack prediction in Chapter 5.

We noticed that there are some cases where an exploit contains an attack payload "wrapped" in another scripting language, for example, a Python script to make a reverse shell connection which uses the Bash *echo* command at the beginning. For these cases, the type of the attack payload is the one with the longest byte sequence. In this example, the type of the particular connection is Python.

It is also important to note whilst the vulnerable servers in our setup used a seven year

Table 3.3: A summary of exploits captured in the BlattaSploit Dataset. The numbers next to the protocols are the number of connections in the application layer protocols..

Number of TCP Connections	5,687
Protocols Included (The number of TCP connections)	HTTP (5515), FTP (6), SMTP (74), POP3 (93)
Payload Types	Javascript, Shellcode, Perl, PHP, Python, Ruby, Bash, SQL

old operating system, the payload carried by the exploit was the identical payload to a more recent exploit would use. For example, both CVE-2019-9670 (disclosed in 2019) and CVE-2012-1495 (disclosed in 2012) can use `generic/shell_bind_tcp` as a payload. The traffic generated by both exploits will still be similar. Therefore, we argue that our dataset still represents recent attacks. Moreover, only attacks that generated exploit payload and actually dealt damage to the victim are kept in the dataset. Attacks that are showing the same behaviour as high-rate attacks are also excluded, making the recorded traffic more representative to low-rate attacks.

3.3 Conclusion

In this chapter, we have discussed existing datasets that contain low-rate attack traffic and their limitations. Some of them have issues with the lack of contemporary low-rate attacks (e.g., DARPA99, KDD99, NSL-KDD, and Gatech). ISCX12 dataset does not state the attack type of its malicious traffic, leading to confusion in separating low-rate attacks from the others. Finally, low-rate attacks traffic in UNSW-NB15 is barely distinguishable from the legitimate traffic. Therefore, we argue that UNSW-NB15 can still be used for evaluation if complemented with a more representative network traffic dataset that contains contemporary low-rate attacks.

We have generated a novel attack dataset, BlattaSploit, which is made up of 81.25% of low-rate attacks disclosed between 2010-2017. Each attack was launched with various combinations of attack payload and encoder, making the generated traffic diverse. Moreover, the dataset also has information on the type of attack, the type of the attack payload, the type of encoder, and the location of the attack payload in the application layer message. To the best of our knowledge, at the time of writing, BlattaSploit is a dataset which includes the widest range of contemporary low-rate attacks, giving the following contribution:

C2: *A comparative study of network traffic datasets to better understand how representative they are to be used for evaluating NIDS and a dataset of low-rate attack traffic with state-of-the art attacks, various payloads, and encoders. Additionally, the dataset contains information about the location of the malicious traffic within the payload. The location is useful to analyse whether an early prediction method can detect the attack before it completes.*

Chapter 4

An unsupervised approach for detecting low-rate attacks in network traffic

4.1 Introduction

New vulnerabilities appear every day, and thus, adversaries always have a new vector to exploit. Furthermore, most of these attacks can be conducted remotely, and the adversary can launch the attack from anywhere in the world and target various application layer protocols. For instance, Cisco Data Center Network Manager has a vulnerability in which an authenticated user can upload a Web Application Resources (WAR) file over HTTP containing malicious scripts and execute the remote command as root [11]. Another example, some versions of PureFTPd may become an entry point for adversaries to exploit Bash Shellshock vulnerability [6] by embedding the remote code in FTP messages. By exploiting these vulnerabilities, adversaries may gain control of our computer without having to be physically present in front of the computer. It increases the possibility of our system being attacked.

As discussed in Chapter 2, there have been several studies on detecting low-rate attacks on network traffic. Some of them extract information from the packet header [113, 34, 60, 32, 42, 48, 96, 33, 102, 28]. These works use information such as Internet Protocol

(IP) addresses, ports, packet size, inter-packet time difference to characterise low-rate attack traffic. Some others analyse the payload of network traffic as the content of a packet is deemed to contain more distinctive information which can be used to identify low-rate attacks [107, 108, 86, 103].

Some approaches, particularly the older ones, were evaluated with old datasets, such as KDD Cup 1999 Dataset (KDD99) [113, 34, 60, 32, 42, 48, 96, 33] and 1999 DARPA Intrusion Detection Evaluation Dataset (DARPA99) [107, 24, 108, 94, 21, 58, 82]. Therefore, their approach may not work well with more recent traffic as both network-based applications and low-rate attacks have evolved over the years. Hadžiosmanović et al. [50] and [112] put more strength in this argument by running previous studies [107, 108, 24, 86] on more recent data and their results show performance degradation.

More recent studies show improvements from older research [111, 82, 51, 97, 109]. They also evaluated their approach with more recent data, either from self-generated traffic or publicly available datasets (i.e., UNB Intrusion Detection Evaluation Dataset 2012 (ISCX12) and UNSW-NB15). Nevertheless, these recent studies only focus on Hypertext Transfer Protocol (HTTP), even though their network traffic representation was protocol-agnostic and could have been used for detecting attacks on other protocols. These two issues brought us to our second research question:

RQ2: *Given recent research has shown that previous work on low rate attack detection across multiple protocols is dated and performance drops on contemporary low rate attacks, how do we improve the performance of low rate attack detection models to deal with evolving cyber attacks on a range of protocols that are increasingly causing damage to corporate networks?*

Machine Learning (ML) algorithms are capable of classifying objects and artefacts based on features exhibited in data and handle various modalities of input. ML has been successfully applied in many domains with a high success rate, such as image classification, natural language processing, speech recognition, and even intrusion de-

tection system. There has been much research on implementing machine learning to address network intrusion detection [100].

There are two main approaches to training a ML model, *supervised* and *unsupervised*. Supervised approaches train the model with samples with known labels. In the case of Network-based Intrusion Detection System (NIDS), the classes may be malicious and legitimate. Thus, the model learns to distinguish legitimate and malicious traffic. However, supervised learning has two known issues, imbalanced data and the requirement to have well-labelled samples. The amount of legitimate traffic is always undoubtedly larger than the malicious one, and this is always the case with every publicly available dataset. If we blindly apply a supervised ML method with imbalanced data, the model may prefer to classify legitimate traffic correctly with the cost of missing the malicious instances. The cost of classifying an attack as legitimate traffic is much higher than the other way around [100]. A common workaround to handle this problem is by adding artificial samples to the underrepresented classes or randomly sampling from the over-represented classes [66].

The lack of malicious samples is another issue with supervised approaches. Supervised learning generally performs better than unsupervised at classifying known attacks [63], but it may struggle to detect attacks which are not similar to those it has seen during the training. In the real world, attack methods are changing all the time and it is extremely difficult to obtain enough recent malicious network traffic to train a supervised ML model. This problem may lead to the ML model becoming 'out of date' and miss more recent attacks.

Unsupervised learning, on the other hand, is more commonly used to draw inference from data without known labels. A task that belongs in this category is anomaly/outlier detection. Anomaly detection is a task where a model is trained over samples of 'normal' data or legitimate traffic in this case and then looks for any new data which deviate from the normal model. This is particularly useful when features/activities that are part of the target class change frequently (i.e., cyber attacks). For instance, PAYL [107]

clustered network traffic based on the 1-gram representations of bytes in its payload. It then calculates the distance between a new incoming packet and the normal model with simplified Mahalanobis distance. An alert is raised when the distance surpasses a threshold value. Perdisci et al. [86] proposed an ensemble of one-class Support Vector Machine (SVM)s, an unsupervised variation of SVMs which is trained with data from one class (i.e., the normal/benign class), to identify shellcode in network traffic. New traffic that is voted by the one-class SVMs as not part of the class is deemed malicious. OCPAD [103] used a similar concept with one-class Naive Bayes to look for malicious network payload. All these approaches do not require malicious traffic samples for training the model, and are therefore likely to perform better in an environment with frequently changing behaviour such as computer networks. This benefit led us to focus on the unsupervised approach in this chapter.

The performance of machine learning algorithms depends heavily on the representation of the given data. This representation, also is referred to as features, are usually manually crafted with some input from human knowledge. A flow in network traffic can be represented by, for example, the number of packets transmitted, the average size of packets, the inter-arrival time between packets, etc. Most publicly available network traffic dataset (e.g., KDD99, ISCX12, UNSW-NB15) includes such information to help researchers designing their NIDS. Using the mentioned representations as features is common in header-based NIDS [78, 28]. However, we argue that this information is not enough to capture the behaviour of low-rate attacks as it contains less distinguishable information than those obtained from within the payload in the case of low-rate attacks [39]. The question is, what would be the best way to represent a network payload?

Manually crafted features from the network payload would be difficult to obtain as there are many application layer protocols with various message format. For instance, HTTP, FTP, and SMTP messages are completely different from each other. It requires a great deal of human effort and time to craft features for each protocol. Therefore,

previous works on payload-based NIDS used a more high-level and abstract representation of data such as frequency of bytes [107, 24, 82, 58], byte sequences [21], or byte subsequences [108, 94, 86, 111, 103, 25]. Abstract features can capture *factors of variation* which explain the observed data better [47]. For instance, different browsers requesting the same web page from a web server send a slightly different HTTP request message. However, these messages would be different from the message sent by adversaries to exploit a vulnerability in the web server. High-level and abstract features may handle the variation of messages sent by different browsers, and the model would still classify them as legitimate traffic.

The aforementioned works then apply a detection method that uses the abstract representation of the network payload to identify attacks. Some of these works employ threshold-based statistical model [107, 108, 94, 82], while others used ML algorithms, e.g., Logistic Regression [111], SVM [86], Hidden Markov Model (HMM) [21]. Despite the fact that these works used abstract representation of network payload, the statistical model and the conventional ML algorithms do not work well with raw-form data or abstract features, particularly when facing a vast amount of data [64]. Therefore, we argue that there is a room for improvement for the detection method. Deep learning, which is also a type of ML algorithms, can deal with the high-level abstraction of data better than the conventional ML algorithms and works well at detecting patterns in large volumes of data. [47]. Although, not all deep learning architectures are compatible with unsupervised learning.

Therefore, in this chapter, we will investigate how well unsupervised Deep Learning (DL) models with payload-based features can identify low-rate attacks in network traffic. We will first discuss which DL architecture can be used for outlier detection. Afterwards, the proposed detection methods are described. Lastly, we show how our proposed method performs on recent datasets and several protocols. To evidence an improvement of our proposed method over the state-of-the-art, we compare the results of our experiment with other NIDSs. The existing works were picked based on their

source code or pseudocode availability and whether their approach is unsupervised. All of these works then were evaluated using recent datasets, UNSW-NB15[80] and BlattaSploit.

To summarise, the contribution discussed in this chapter is:

C3: *Unsupervised deep learning models to identify low rate attacks in network traffic, putting aside the requirement to provide malicious samples for the training data. The proposed approach offers an improvement in detection rate at least 12.04% from the previous works.*

4.2 The Basics of Deep Learning for Outlier Detection

Deep Learning (DL) can be used in supervised or unsupervised mode, depending on the architecture used. As this chapter focuses on the latter, we will cover which architectures are compatible with outlier detection and suitable for our case. Before we go into that, we will first introduce the basic concept of deep learning.

Before DL rose to prominence, Artifical Neural Network (ANN)s were the state of the art- referred to as feedforward neural networks or multilayer perceptron. An ANN model consists of multiple layers in which information flows from the input layer to the output layer with several computations performed in the middle layers. The middle layer is referred to as the **hidden layer**.

Each hidden layer contains several neurons that can be activated or not depending on the output of the **activation function** of the neuron. It allows important information to be forwarded to make the decision, while unimportant information is blocked. A neuron accepts a vector of input x and does a linear transformation $y = W*x+b$, where W is the weight, and b is a bias. y then is processed by the activation function. During the training phase, an ANN model will compare the difference between its output and the target. The difference is then propagated back to the previous layers. Thus they

can update the weights and biases of their neurons accordingly, hence improving the quality of the model. This method is called **backpropagation**.

At the beginning of their appearance, ANNs were not preferable due to their complexity and lack of powerful hardware. The recent development of graphics cards has brought a new light to solve that problem. Newer graphic cards enable us to speed up the ANN model computation. They decrease the training time and increase the number of calculation a computer can perform in parallel. This recent development allows researchers to add more hidden layers. It enables the model to solve complex non-linear problems.

Inspired by how our brains work, DL has a capability of recognising more abstract representation of data. For instance, a DL model can classify images by analysing the pixel values [47]. As another example, a DL model capable of classifying documents by looking at the words/letters inside them. Therefore, it is intriguing to see whether DL with unsupervised learning can help in detecting low-rate attacks by analysing more abstract representation of network traffic.

DL includes feedforward neural networks with *more than one* hidden layers. At the time of this writing, DL has many architectures [47]. In the next few sections, we will discuss some well-known DL architectures in more detail, which can be used for outlier detection, such as **Recurrent Neural Networks** and **Autoencoders**.

4.2.1 Recurrent Neural Networks

Recurrent Neural Network (RNN)s were designed for processing sequential data [47]. An RNN model takes an input sequence denoted by $X = x_1, x_2, ..., x_n$ where n is the sequence length and x_i is a feature vector for $1 \leq i \leq n$. The unfolded view of a RNN is shown in Figure 4.1. Unlike the feedforward Neural Network (NN), the hidden neuron/unit has a feedback loop where the output at a time step is passed to the next time step calculation carrying information from the previous time step. Therefore,

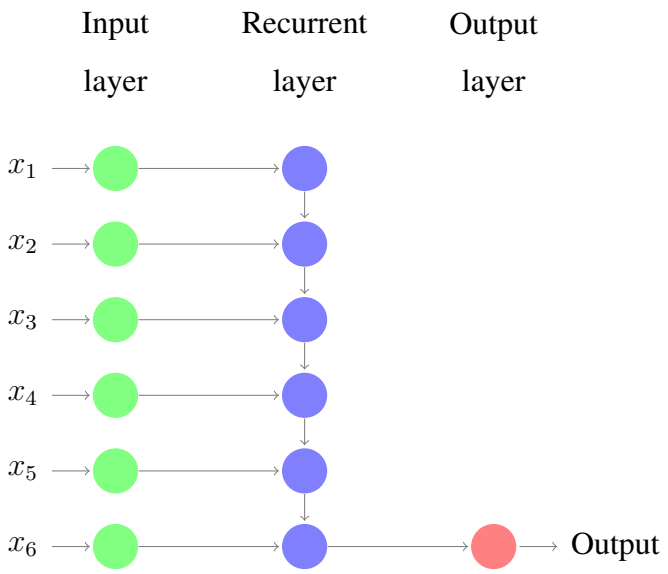

Figure 4.1: An unfolded view of a Recurrent Neural Network which processes a sequence of vectors and outputs a value. This model is commonly used for document classification in which the vectors usually represent words, and the output is the class of the document..

it is said that the hidden neuron shares parameters. It makes RNNs able to learn the property of a sequence better.

RNNs are widely used in natural language processing area due to their ability to work well with sequential/temporal data. They can predict the next item in a sequence given a sequence of previous items. However, vanilla RNNs are known to have a **vanishing gradient problem**, which usually appears when they work with a long sequence. It is a problem that occurs when the gradient used to update the neuron weight becomes extremely small, preventing the neuron from updating its weight. Gated RNNs can avoid this problem by adding gate(s) to a neuron, thus selecting which information to be passed to the next time step calculation. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are a further development of RNNs that utilise this approach.

Existing works on NIDS utilising an RNN model, or its further development such as

LSTM and GRU, are usually supervised [92, 52, 69]; although some research in other areas has shown to be incorporating RNNs for outlier detection. Malhotra et al. [73] proposed an LSTM model that can identify outliers by looking at whether the model can correctly predict the next item in a sequence. In that research, they experimented with sequences of ECG, valve, power consumption, and multi-sensor engine data. The model is trained to predict the next element in the sequence, so that it remembers the temporal patterns of the training data. Therefore, when the model sees a new and unknown sequence, the prediction is expected to be make a lot of mistakes. The sequence is then deemed anomalous. Feng et al. [43] proposed Industrial Control System (ICS) attack detection with an unsupervised LSTM similar to what Malhotra et al. [73]'s approach, but their work cannot be used to detect attacks on TCP/IP-based protocols as their features were specifically crafted for Modbus traffic. In the following section, we will describe more how this approach can identify low-rate attacks in network traffic but before that we will discuss Autoencoders, another DL architecture which can be utilised as an outlier detection.

4.2.2 Autoencoders

The usual feedforward neural networks map a set of input features to a target output or a class. They typically have a smaller number of neurons as the layer goes deeper because each layer extracts more abstract representations and passes them so that the model will get a more concrete representation and can make a better decision [47]. For instance, in the first layer of an image recognition model, the neurons may contain information about lines. The second layer may recognise edges. Then the third layer may see the shape which will be used to determine what image the picture has. In other words, feedforward neural networks compress and extract information.

In contrast to the purpose of classifying objects with feedforward neural networks, an autoencoder is typically used to learn a representation of a set of data by encoding/compressing the input then decoding/decompressing in the hidden layers. It is

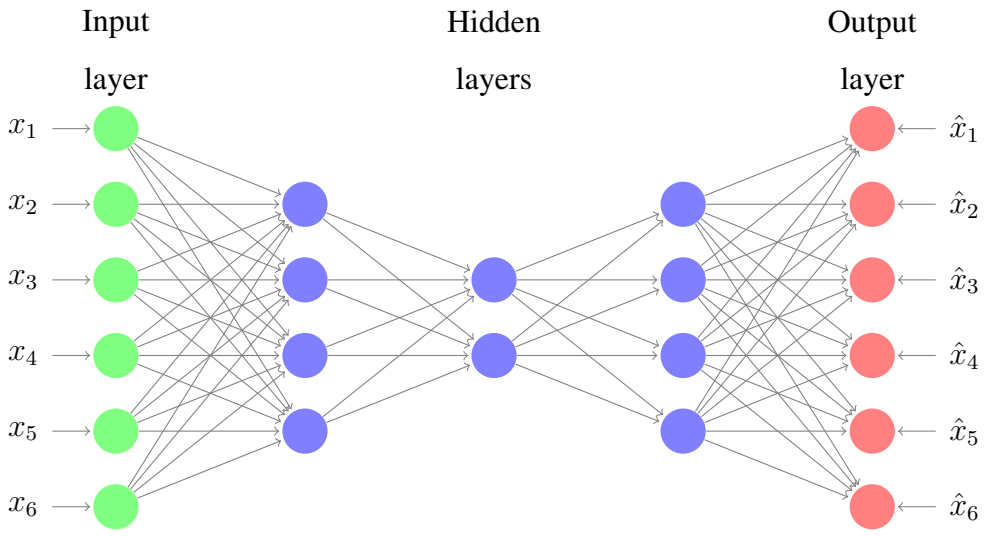

Figure 4.2: An example of an Autoencoder

essentially trained to attempt to copy its input values to its output [47]. Hence it has the same number of input and output neurons, as shown in Figure 4.2. The encoding-decoding process removes or blurs noises in the data. Additionally, an autoencoder can recognise outliers since unseen data would not be correctly reconstructed, resulting in a huge difference between the input and output values [53]. This difference is referred to as a *reconstruction error*.

With autoencoders' ability to detect outliers/unseen data, it is possible to use them to identify anomalous behaviour in network traffic. For example, Kitsune [78] takes packet size, packet count, and packet jitter as a feature set to identify Probe, Man in the Middle (MitM), and Denial of Services (DoS) attacks. They argued that attack traffic would exhibit different values of those features. However, Kitsune [78] does not focus on detecting low-rate attacks nor analysing payload. It remains a question of whether the approach can work on low-rate attacks.

In the following section, we will discuss in more detail how RNN and Autoencoder-based approaches are incorporated separately to our proposed method for detecting low-rate attacks. We will then investigate the performance of both approaches and find out which unsupervised approach is more suitable for identifying low-rate attacks in

Section 4.4.

4.3 Low-rate Attack Detection Methodology

This section details our two methods to detect low-rate attacks in network traffic, as described in Figure 4.3. The first method utilises an RNN-based model for outlier detection, while the second employs an Autoencoder. The first step in these two approaches, the network traffic preprocessing, is the same. The step reconstructs network packets to an application layer message. It is conducted to ensure that our models can see the full picture of the payload which we argue will result in better accuracy in comparison to approaches that process network packets individually [107, 103]. Each detection method has its feature extraction method as both require a different representation of network payload. The RNN-based method needs a sequence of bytes from the payload, while the autoencoder-based method involves a set of byte frequencies.

Although RNN and Autoencoders have been used as an outlier detection method in other areas, there has been no research that utilises them to identify low-rate attacks by reading the network traffic representation that we proposed (i.e., byte sequence and byte frequencies). Moreover, we also proposed two statistical approaches to help the model decide whether the output of the RNN or Autoencoder belongs to the normal model.

4.3.1 Network Traffic Preprocessing

Application layer messages are usually longer than the maximum size of an IP packet. They are therefore needed to be segmented into multiple IP packets and sent individually, sometimes they arrive in order, sometimes not. A payload-based NIDS that reads the application layer message would have a more complete picture of the message being transmitted and thus more accurately detect the attack. Therefore, in our

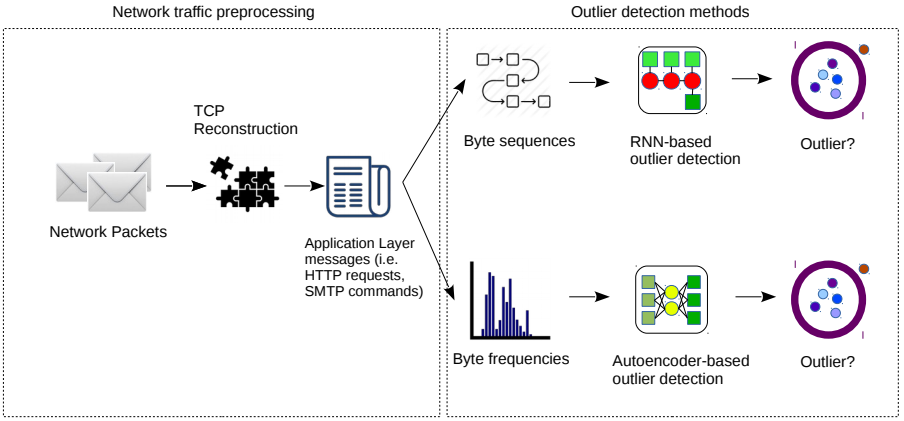

Figure 4.3: The overview of the proposed methods. The first step is applicable to both methods, which is reconstruct application layer messages from network packets. Each method has its own set of features but both aim at identifying outliers in the network traffic and mark them as malicious..

approaches, when a network packet arrives, it is not directly processed by the outlier detector. The packet is put in a queue buffer instead and then, together with other packets of the same flow (identified by IP addresses, ports), are reassembled to an application layer message. We implemented the application layer message reassembly from scratch by following the RFC 793 standard [4], as there is a bug in the PyNIDS[79] library. PyNIDS includes arbitrary bytes in the reconstructed messages which would confuse our RNN and Autoencoders models. PyNIDS is a wrapper for libNIDS which was written in C and has not been maintained since eight years ago, therefore fixing the bug would have taken a long time. We verify that our implementation is correct by comparing the reconstructed application layers messages with the Wireshark *Follow TCP Stream* feature.

A typical TCP connection is finished when a FIN packet is received. When the packet is received, the connection is marked as ready and can be examined by the outlier detector. However, connections may be interrupted for various reasons. Thus we also

monitor how long a TCP connection has been in the buffer. If it has been in the buffer longer than a predefined timeout value, the connection is deemed as interrupted and marked as ready to be processed by the outlier detector. Finally, messages of each application layer protocol are grouped as each protocol is analysed independently. In other words, we will have a model for each protocol. These models is trained with the same method, but a different training set. Therefore, our proposed methods can detect low-rate attacks on multiple protocols.

4.3.2 Outlier Detection Models

Section 4.2 has explained the basic concept of RNN and Autoencoders for outlier detection. In this section, we are more focused on how we build the deep learning structure and how we apply the model to our data, application layer messages. This section also formulates the way we transform the application layer message to a feature set, train the model, and feed it to the algorithm.

As our proposed methods are unsupervised, we first collected legitimate traffic from the dataset, reconstruct the application layer messages from it, and train the models on those data. In the training phase, it is anticipated that the models learn the pattern of network payload in legitimate traffic. Afterwards, the models will be able to identify any deviations from the normal pattern and marked them as malicious.

We formulate M_p as a set of messages m of an application layer protocol p (i.e., HTTP, Simple Mail Transfer Protocol (SMTP)). Both proposed models (RNN-OD and AE-OD) work with M_p, but each model has its network payload representation as inputs to the model as it has a different method to detect outliers. The RNN-based model uses a sequence of byte values to represent an m_p, while the Autoencoder-based model uses byte frequency as the representation. Both approaches are explained further in the following sections.

Outlier Detection with Recurrent Neural Networks

Our RNN-based outlier detection model (RNN-OD) works based on the principle that low-rate attacks would have unusual byte sequence. RNN-OD takes overlapping subsequences of bytes with a particular length. It is then trained to predict what the next byte of each subsequence is. The training process enables the RNN-OD to learn the subsequence of bytes that usually appear in legitimate traffic. An unseen low-rate attack is expected to make the RNN model output incorrect next-byte predictions as it may contain a sequence of bytes that rarely appears in legitimate traffic and is unknown to the model.

Formally, we define the sequence $B = \{b_1, b_2, \ldots, b_l\}$ of length l as a sequence of bytes, where each byte b_i is a byte in an application layer message m_p with the length of l. We choose some length of n, such that $n < l$.

Then we define the set S as in equation 4.1 where each s_i is a subsequence of length $n + 1$ of B.

$$S = \{s_i | s_i = (b_i, b_{i+1}, b_{i+2}, \ldots, b_{i+n+1}) \forall 1 <= i <= l - n + 2\} \tag{4.1}$$

We then split each subsequence s_i into x and y where x is the input to our RNN-OD model and y is the target output:

$$x = \{s_k \mid 0 \leq k < n\} \tag{4.2}$$

$$y = s_n \tag{4.3}$$

Hence, we train a function F_p, where function F_p is a recurrent neural network-based classifier to predict the next item $\hat{y}$ from a sequence x, over the set S (as S is now defined as the set of all sequences s_i) in protocol p in the training phase. The function

F_p is then optimised with backpropagation so that the predicted output $\hat{y}$ equal to the actual value of y. For instance, during the training phase, F_{HTTP} reads all HTTP messages, F_{FTP} takes File Transfer Protocol (FTP) messages, and F_{SMTP} deals with SMTP messages. After obtaining the messages, each of those functions will collect a set of subsequences s from them.

Stating that our RNN classifier as $F_p(x)$ would be an oversimplification as there are many operations involved in the function. Thus, we formulate our function $F_p(x)$ in more detail as:

$$F_p(x) = argmax(SF(R(E(x)))) \tag{4.4}$$

Where $E(x)$ is a function that transforms each x to a vector with a specific dimension. It can also be referred to as Embedding layer. Function R is the recurrent layer which takes embedded vectors as inputs and outputs an intermediate vector value which is processed by a linear softmax function SF. The output of the Softmax function SF is the probability of bytes being the next item in the sequence. Argmax function returns the item with the highest probability.

Up to this point we have referred to the use of RNNs. In fact, we use LSTM and GRU - types of RNN which work better with longer sequences as they can filter which information should be passed onto the next time step calculation by incorporating *gates* [54, 35]. In general, the purpose of R function does not change, neither does the dimension of inputs and outputs.

To measure how far the new incoming traffic from the normal model, RNN-OD needs to calculate the anomaly score. We first obtain the anomaly score of each application layer messages seen in the training phase to calculate a threshold that will be used in the detection phase. Any application layer message which has an anomaly score above the threshold is deemed malicious.

As illustrated in Figure 4.4, our RNN-OD model works by taking a subsequence x and

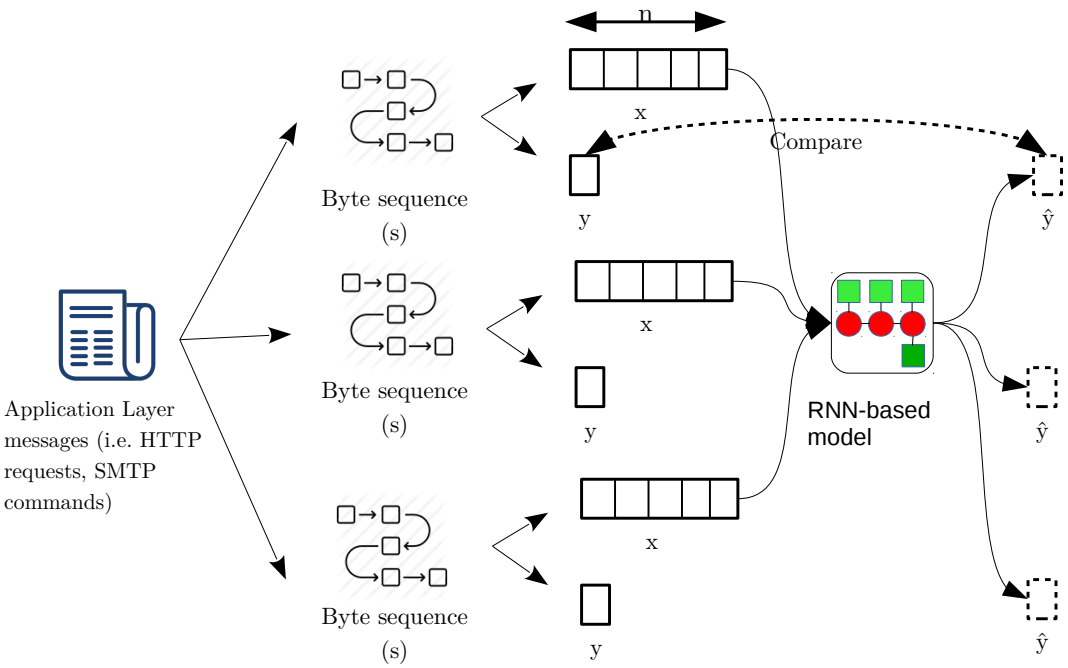

Figure 4.4: A detailed view of how RNN-OD works. The RNN model accepts byte sequences of application layer message and tries to predict the next byte of the subsequence. The RNN model is expected to make many prediction mistakes when facing a low-rate attack as it supposedly has never been seen during the training phase..

throwing the predicted next item $\hat{y}$. The anomaly score is essentially the difference between the predicted and actual outputs, yet this difference can be measured discretely or continuously. Therefore, in this part, we propose two methods to calculate the anomaly score a_p of an application layer message m_p, namely binary and floating score.

Binary anomaly score is calculated by counting the number of correct prediction the

model makes. Let $\hat{Y}$ be a set of predicted items $\hat{y}$s:

$$\hat{Y} = \{\hat{y}_i | 0 \leq i < (l - n)\} \tag{4.5}$$

Then the binary anomaly score a_p^{binary} of message m_p is defined as follows:

$$a_p^{binary} = \frac{\sum_{i=0}^{l-n} v_i}{l} \begin{cases} v_i = 1, & \hat{y}_i = y_i \\ v_i = 0, & otherwise \end{cases} \tag{4.6}$$

Floating anomaly score is calculated by accumulating the difference of the predicted probability of $\hat{y}$, which can be obtained from the output of the Softmax function, and the actual probability of y. Therefore, if $Prob(y)$ is the probability of y, the floating anomaly score a_p^{float} of message m_p is defined as:

$$a_p^{float} = \frac{\sum_{i=0}^{l-n} (Prob(\hat{y} - Prob(y))^2}{l} \tag{4.7}$$

In the detection phase, we first obtain a set of subsequence s from the application layer message as described by Equation 4.1. Each subsequence s is split into x and y as described by Equation 4.2 and 4.3 respectively. Subsequently, the anomaly scores (i.e., binary and floating) are calculated. An alert is raised when the anomaly score surpasses the threshold (the method of defining the threshold is explained in detail in Section 4.4.3).

Outlier Detection with Autoencoders

As explained in Section 4.2.2, an autoencoder is a model which attempts to copy the input to its output. This behaviour makes an autoencoder able to learn the data representation. In other words, an autoencoder is trained to memorise the pattern of training data. In this case, we aim to develop a model which remembers the pattern of legitimate traffic that we refer to as AE-OD. Figure 4.5 generally shows how AE-OD works.

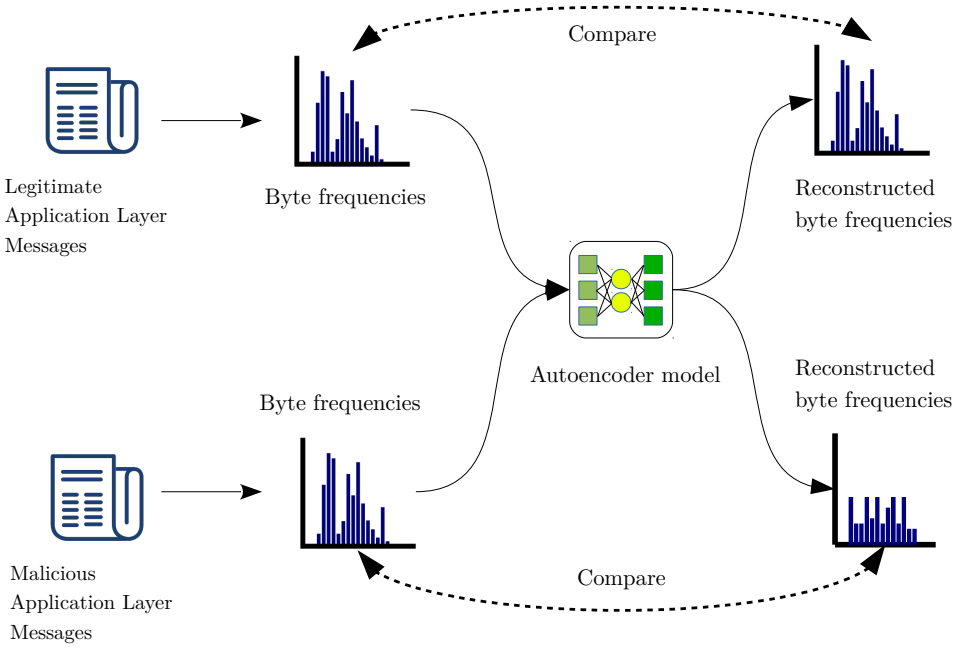

Figure 4.5: A detailed view of how AE-OD works. Byte frequencies of an application layer message is fed into the Autoencoder to be reconstructed. Legitimate application layer messages would have reconstructed byte frequencies that are similar to the inputs, while low-rate attacks would have reconstructed byte frequencies which are far from the inputs, resulting in higher anomaly scores..

We represent application layer messages in the traffic as byte frequencies as low-rate attacks would exhibit a different distribution of byte frequencies. We could have used byte sequence as the data representation, as we do with RNN-OD, but Autoencoder is not designed to work with sequential data with variable length. As a byte value varies from 0 to 255, byte frequencies are the numbers of occurrence of every possible byte value in an application layer message divided by the message length. The division is needed so that the frequency of a byte is relative to the length of the message. Therefore, the model will not be biased toward either short or long messages. Similar to RNN-OD, AE-OD analyses legitimate traffic in the training phase.

Let m_p denote a legitimate application layer message transmitted with protocol p. We then define $X = \{x_i \mid 0 \leq i \leq 255\}$ as a set of byte frequencies of m_p. The vector of byte frequencies X is the input to the Autoencoder. An Autoencoder model is essentially a non-linear function G_p which collects an input vector X from a protocol p message and transform it to $\hat{X}$. The function can be defined as follows:

$$\hat{X} = G_p(X) \tag{4.8}$$

, where $\hat{X} = \{\hat{x}_i \mid 0 \leq i \leq 255\}$. Function G_p is then optimised with backpropagation so that $\hat{X}$ is as close as possible to X, and similar to the RNN-OD model, G_p is a stack of hidden neural network layers, although they are not recurrent in this case.

In the training phase, AE-OD reads legitimate application layer messages and groups them by the protocol used. For each protocol, it trains an autoencoder model to reconstruct the byte frequencies of the application layer messages.

As is the case with RNN-OD, it is needed to measure the distance between the normal model and the new incoming traffic. The anomaly score for each protocol in AE-OD a_p^{ae} is obtained from the mean squared of the sum difference between each element in X and $\hat{X}$ respectively. As mentioned in Section 4.2.2, it can also be referred as a *reconstruction error*, but from this point onwards we will keep the term anomaly score for brevity. We define the anomaly score of AE-OD as follows:

$$a^{ae} = \frac{1}{256} \sum_{i=0}^{255} (x_i - x_i')^2 \tag{4.9}$$

In the detection phase, AE-OD analyses application layer messages, obtains the byte frequencies, feeds them to the autoencoder model, and calculates the anomaly score. Every application layer message with an anomaly score higher than the threshold is deemed malicious. The threshold, similar to RNN-OD, can be defined manually. However, we propose to use a statistical approach to define the threshold so that it is also learned from the training data (see Section 4.4.3).

4.4 Experiments and Results

This section details our experiments with both RNN-OD and AE-OD. We first detail our experimental setup, including the parameters used, the previous works to be compared with, and the metrics used to measure the performance.

We then describe the datasets used to evaluate our proposed methods. In summary, UNSW-NB15 is used as it contains more recent legitimate traffic and several low-rate attacks. We also assessed the methods with our generated low-rate attack dataset, BlattaSploit to see how good our proposed methods in detecting contemporary low-rate attacks.

The approach to statistically define a threshold is also discussed later on in this section. The training data is analysed to see what the most suitable approach to pick a threshold value as it would affect the performance of our proposed methods.

Lastly, we will show the results of our experiments. To validate our results, we also compared them with the result of previous works. Our extensive experiments show improvements in detection rate with the cost of increasing false positive.

4.4.1 Experimental Setup

We implemented the proposed methods using Python 2.7.12 with Keras 2.0.6 [1], Tensorflow 1.2.1 with GPU support, and pcapy 0.10.8. All experiments were conducted on a personal computer with Intel Core i7-6700 @3.40 GHz, 16 GB RAM, and NVIDIA GeForce GT-730.

The hyperparameters values of RNN-OD is shown in Table 4.1 and of AE-OD is shown in Table 4.2. There are a couple of other parameters that we explored to see how they affect the performance, i.e., the length of subsequence n for RNN-OD and number of neurons in the hidden layers of AE-OD. The options for each parameter is shown in Table 4.3.

Table 4.1: RNN-OD hyperparameters configuration

Hyperparameters	Value(s)
Recurrent Output Dimension	32
Number of Epochs	10
Activation Functions in Hidden Layer(s)	Hyperbolic Tangent
Activation Functions in Output Layer	Sigmoid
Dropout	0.2
Optimiser	adadelta
Loss Function	Categorical Crossentropy

We set the activation function of the recurrent layer to hyperbolic tangent as it converges faster than Sigmoid [18] and works better in recurrent layers than ReLU. While in AE-OD, we use ReLU as the activation function for the hidden layers as they are not recurrent, and it gives faster and efficient training on large datasets than Sigmoid [45]. As an optimiser, we chose Adadelta for both models as it has less computational overhead than the vanilla stochastic gradient descent[114]. Then, we set the number of epochs to ten as the loss value starts being steady after ten epochs. To prevent overfitting, we also add Dropout layers with 0.2 probability. Finally, to provide reproducibility of the result, the complete source code of our implementation is available on https://github.com/bazz-066/neuralnetwork-AD (RNN-OD) and https://github.com/bazz-066/aeids-py (AE-OD).

We compared our proposed methods with PAYL [107], Kitsune [78], Decanter [25], OCPAD [103] and Wang et al. [110]'s. Generally, we picked these works to evaluate due to their code or binaries availability and their unsupervised approach. Kitsune [78] and Wang et al. [110]'s are header-based NIDS, while PAYL, Decanter, and OCPAD are payload-based NIDS. The header-based and payload-based NIDSs were included

Table 4.2: Autoencoder's hyperparameters configuration

Hyperparameters	Value(s)
Number of Hidden Layer(s)	1;3;5
Activation Functions in Hidden Layer(s)	ReLU
Activation Functions in Output Layer	Sigmoid
Dropout	0.2
Optimiser	adadelta
Loss Function	Binary Crossentropy
Number of Epochs	10

Table 4.3: Changeable Parameters in RNN-OD and AE-OD

Hyperparameters	Value(s)
Length of subsequence n	3; 5; 7
Number of Neurons in Hidden Layer(s)	(200); (200, 100, 200); (200, 100, 50, 100, 200)
Recurrent Layer Type	LSTM, GRU

in the experiments so that we can also compare the results of using features from the packet header and network payload. McPAD [86] was going to be included, but its packet capture library was unable to read PCAP files captured with Linux cooked-mode capture(SLL), which was used by both UNSW-NB15 and BlattaSploit to capture the network traffic.

We measure our proposed system's performance and previous works with a combination of detection rate and false-positive rate as they are more intuitive than accuracy. The amount of legitimate traffic is always much greater than malicious traffic. Accuracy metric only measures the number of correct prediction regardless of the type of the

actual data. If an evaluation dataset contained 99% of legitimate traffic, by randomly guessing all incoming traffic as legitimate, the method would get 99% accuracy while it missed all the attacks.

Detection rate (DR) and false-positive rate (FPR) is calculated as in equation 4.10 and 4.11. True positive (TP) is the number of detected malicious connections. False-positive (FP) is the number of legitimate connections which are considered malicious. True negative (TN) is the number of legitimate connections which are deemed as legitimate. False-negative (FN) is the number of malicious connections that go undetected.

$$DR = \frac{TP}{TP + FN} \tag{4.10}$$

$$FPR = \frac{FP}{TN + FP} \tag{4.11}$$

As we have two metrics to measure the performance, it is crucial to find the balance between these two metrics as increasing the detection rate could also come with the increase of the false-positive rate. This balance varies between cases and depends on the cost of undetected attacks. When the attack has severe effect to the system, which is the case for low-rate attack detection, it would be better to prioritise the detection rate over the false-positive rate as the cost of false-positives is less impactful [100]. As the number of legitimate and malicious samples is highly imbalanced (see Section 4.4.2) and to emphasize lower false negatives, we use another metric, F_2-score, which is derived from F_β-score with β is set to 2. This metric finds the balance between the detection rate and false-positive rate. Therefore, it helps us to conclude when both detection rate and false-positive rate increase. F_2-score is calculated as in equation 4.12.

$$F_2 = \frac{(1 + 2^2) \times TP}{(1 + 2^2) \times TP + (2^2) \times FN + FP} \tag{4.12}$$

4.4.2 Datasets

As previously mentioned in Chapter 3, publicly available network traffic datasets suffer from several issues. DARPA99 dataset is obsolete and does not represent contemporary traffic, both legitimate and malicious. ISCX12 dataset does not have information about attack types which makes it impossible to collect low-rate attacks from the dataset as it contains several high-rate attacks (i.e., DoS and Brute force attacks). UNSW-NB15 dataset is not perfect as well. Nevertheless, it is the most recent dataset that contains contemporary and representative legitimate traffic. Therefore, we still use it to evaluate our methods.

UNSW-NB15 dataset comprises two days of captured network traffic, the first one was captured on 22 January 2015 (UNSW-JAN) and the other traffic was collected on 17 February 2015 (UNSW-FEB). The difference between those two days is the amount of traffic since the latter has roughly ten times more traffic. Both of them contain ten classes of traffic, i.e. Normal, Analysis, Backdoors, DoS, Exploits, Fuzzers, Generic, Reconnaissance, Shellcode, and Worms. The explanation of those classes is shown in Table 3.1. To detect low-rate attacks, we only used Normal, Backdoors, Exploits, and Worms classes in the experiments.

We also used our BlattaSploit dataset, which contains various low-rate attacks to evaluate RNN-OD and AE-OD. Since the traffic is entirely different from the one in UNSW-NB15, we can also see how our models perform with data that have never been seen before.

In our experiments, the training set was obtained from legitimate traffic in UNSW-JAN. The testing set consists of legitimate traffic in UNSW-FEB, malicious HTTP and SMTP traffic from both UNSW-JAN and UNSW-FEB, and malicious traffic in BlattaSploit. We only tested all methods on HTTP, FTP, and SMTP, since they are the protocols with the highest proportion of traffic in the UNSW-NB15 dataset.

UNSW-NB15 dataset has 17,041 HTTP, 1,232 FTP, and 4,631 SMTP attacks. Blat-

taSploit contains 5515 HTTP, 9 FTP, and 74 SMTP attacks. We will analyse the false positive (legitimate traffic classified as malicious) with legitimate traffic in UNSW-FEB as traffic from UNSW-JAN has been used to train the model. There are 153,718 legitimate HTTP, FTP, and SMTP connections in UNSW-FEB.

4.4.3 Defining Threshold

The proposed methods are trained on the training set (i.e., legitimate traffic of UNSW-JAN). After the training finished, the same set of samples is fed into the model to obtain the anomaly scores to calculate the threshold. The threshold value is calculated based on this set of anomaly scores. RNN-OD and AE-OD have its own threshold value but the method to calculate it is the same.

The basic idea is that the anomaly score of a low-rate attack would be far from the average anomaly scores of legitimate traffic in the training set. Low rate attacks are supposedly exhibit different byte sequence or frequencies and have never been seen by the model. Therefore, RNN-OD would struggle predicting the next item in the sequence and AE-OD would have a difficult time reconstructing low-rate attacks byte frequencies. The question is, how do we obtain the average anomaly score? A traditional statistical method utilises mean (μ) and standard deviation (σ). This approach works with an assumption that outliers usually fall outside this range: $\mu-2*\sigma > a_p > \mu+2*\sigma$ or $\mu - 3 * \sigma > v > \mu + 3 * \sigma$. However, this approach requires the data to be normally distributed. Therefore, to verify that our data is normally distributed, we performed a series of statistical tests.

As mentioned previously, RNN-OD and AE-OD were trained with legitimate traffic of UNSW-JAN, and the same set is fed again into the model to calculate the anomaly scores. We then investigated whether the anomaly scores produced by RNN-OD and AE-OD were normally distributed. We first plot the histogram of the anomaly scores. Figure 4.6, 4.7, and 4.8 show the histogram of anomaly scores generated by AE-OD,

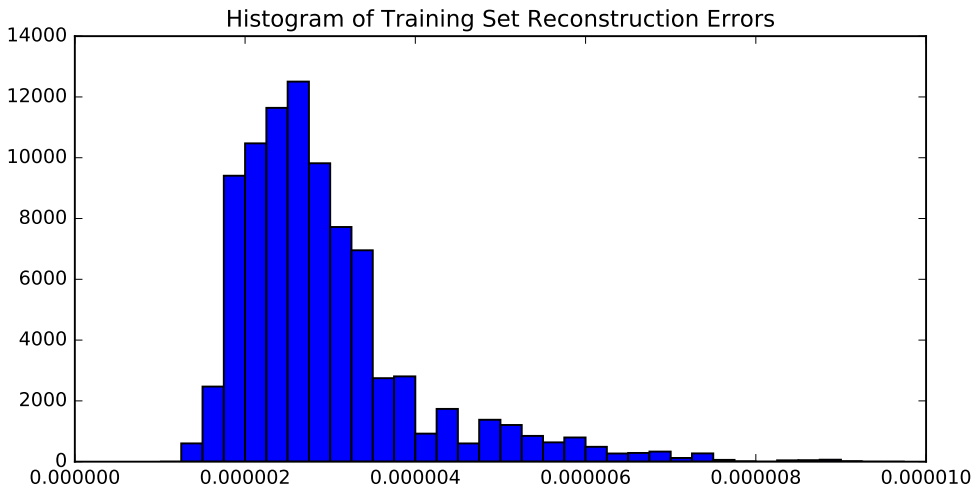

Figure 4.6: Histogram of anomaly scores generated by running AE-OD over the training set. It shows that the reconstruction errors of the training set are skewed.

RNN-OD with binary anomaly score, and RNN-OD with floating anomaly score respectively. These images exhibit similar pattern, the anomaly scores are highly skewed and have multiple peaks.

To ensure that the anomaly scores of the training set are not normally distributed, we also ran a normality test [38, 83]. We obtained a zero p-value for each set of anomaly scores, which means the null hypothesis is rejected, and the data are *not* normally distributed. Thus, utilising mean and standard deviation to find the threshold is not suitable for identifying outliers.

Diez et al. [40] suggest interquartile range (IQR) to detect outliers. It is supposedly more robust as it relies on the median (m), which is less affected by outliers in the data. However, this approach is still not suitable for skewed data. Hence, we propose to use a modified IQR approach for skewed data proposed by Hubert and Vandervieren [55]. It uses *medcouple* (MC) to measure the skewness of data.

Let F be the data, $MC(F)$ is medcouple of sorted F where $x_1 < x_2 < x_3 < ... < x_n$.

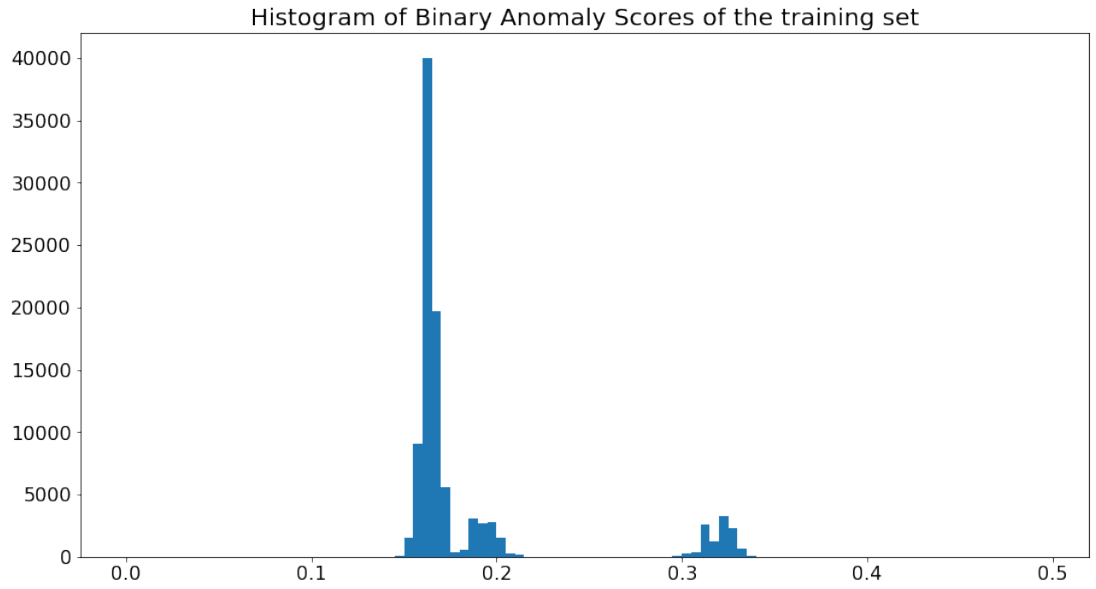

Figure 4.7: Histogram of anomaly scores generated by running AE-OD over the training set. It shows that the reconstruction errors of the training set are skewed.

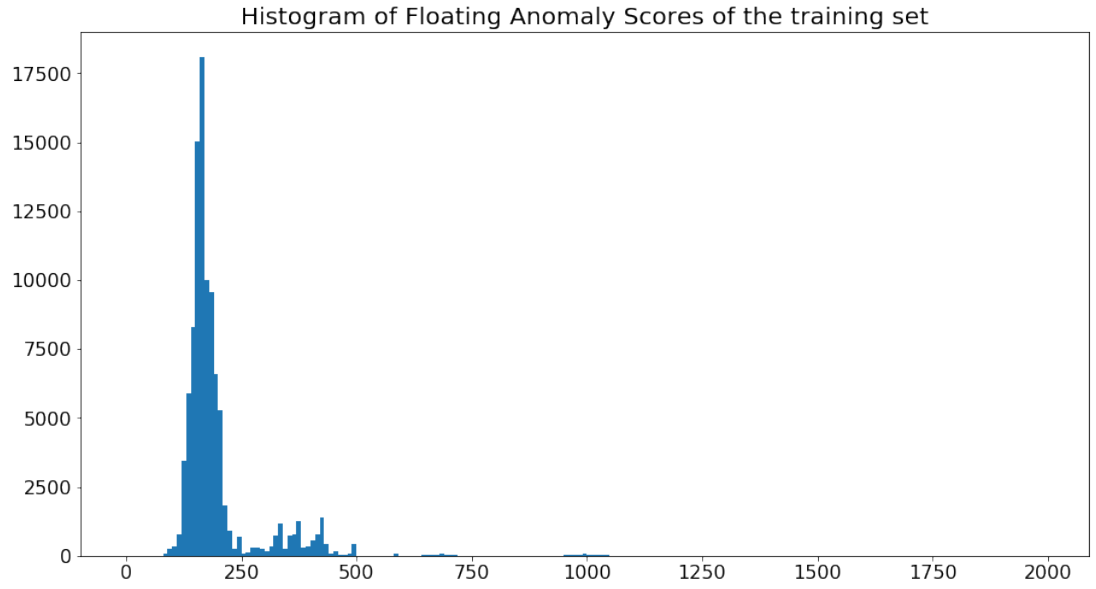

Figure 4.8: Histogram of floating anomaly scores generated by running RNN-OD over the training set. It shows that the reconstruction errors of the training set are skewed.

$MC(F)$ as defined as in equation 4.13. x_i and x_j are sampled independently.

$$MC(F) = \underset{x_i<m<x_j}{\text{median}} h(x_i, x_j) \tag{4.13}$$

$$h(x_i, x_j) = \frac{(x_j - m) - (m - x_i)}{x_j - x_i} \tag{4.14}$$

Afterwards, let Q_3 be the 3^{rd} quartile of the data, then T_{IQR}, the threshold of the modified IQR method is defined as in equations 4.15 and 4.16.

$$T_{IQR} = Q_3 + e^{3*MC} * 1.5 * IQR, \text{ if } MC \geq 0 \tag{4.15}$$

$$T_{IQR} = Q_3 + e^{4*MC} * 1.5 * IQR, \text{ if } MC < 0 \tag{4.16}$$

Another method of detecting outlier is by using modified Z-score, which works based on median absolute deviation [56]. The authors argue that median and median absolute deviation are robust measures of central tendency and dispersion, respectively. However, the way it applies in this chapter is different from the T_IQR approach.

We first need to calculate the Z-score of the reconstruction error. Let $E = \{e_i | i < length(F)\}$ be a set of anomaly scores obtained from the training data. Median absolute deviation MAD can be calculated as in equation 4.17. Thus the Z-score z of particular E_i is calculated as in equation 4.18 [56]. When using this approach, a message is considered as malicious when the z is greater than 3.5 [56].

$$MAD = \text{median}(\{(|e_i - \text{median}(E)|), e_i \in E, 0 < i < n\}) \tag{4.17}$$

$$z = \frac{0.6745 * (|e - \text{median}(R)|)}{MAD} \tag{4.18}$$

4.4.4 Results Discussion

In summary, we have six different methods, based on Autoencoders and RNN, to evaluate. RNN-OD and AE-OD both have a set of changeable parameters (see Table 4.3), but for brevity, Table 4.4 only shows the best configuration, in terms of the combination of detection rate and false positive rate, we have found during the experiments. The complete results of the experiment can be seen in Appendix B.

In this section, the number of hidden layers in AE-OD was set to three with 200, 100, and 200 neurons respectively. RNN-OD was configured to use LSTM, and the length of subsequence was set to three.

Table 4.4 shows the result of our experiments. All methods were trained on the same training set (i.e., legitimate traffic in UNSW-JAN). We measure the experiments with three metrics. DR-UNSW shows the method detection rate of low-rate attacks in UNSW-NB15 dataset. DR-BS shows the detection rate of malicious traffic in BlattaSploit dataset. And FPR-UNSW measures the number of legitimate traffic in UNSW-FEB that was detected as malicious. There is no FPR for BlattaSploit dataset as it only contains malicious traffic. We then provide the F_2-score of each method which considers the result of both datasets. We summed the detection rate of both UNSW-NB15 and BlattaSploit dataset and used the FPR of UNSW-NB15 to obtain the F_2-score. As all discussed methods merely raise alerts to the administrator, none of them prevents any traffic from going through. False positives will not cause important messages to be dropped. Therefore, the cost of classifying an attack as benign is greater than deeming a legitimate connection as malicious. The F_2-score helps us to conclude when both detection rate and false-positive rate increase. It emphasizes the detection rate without neglecting the false-positive rate.

In general, RNN-OD with binary anomaly score and T_{IQR} threshold method performed the best among all methods. It achieved a detection rate higher than 99% for both UNSW-NB15 and BlattaSploit datasets while maintaining relatively low FPR (3.57%).

Table 4.4: A result comparison between RNN-OD, AE-OD, and previous works. DR-UNSW shows the method's detection rate on the UNSW-NB15 dataset. DR-BS shows the method's detection rate on the BlattaSploit dataset. F_2 are calculated by including the detection rate of both UNSW-NB15 and BlattaSploit datasets, as well as the false positive rate of legitimate traffic in UNSW-NB15 dataset.

Method	DR-UNSW (%)	DR-BS (%)	FPR-UNSW (%)	F_2
AE-OD (T_{IQR} Threshold)	51.55	96.83	0.89	0.67
AE-OD (Z-Score Threshold)	100	100	23.61	0.74
RNN-OD (Binary anomaly score & T_{IQR} Threshold)	**99.13**	**99.97**	**3.57**	**0.95**
RNN-OD (Binary anomaly score & Z-Score Threshold)	0	0	0	0
RNN-OD (Floating anomaly score & T_{IQR} Threshold)	34.24	58.07	1.12	0.41
RNN-OD (Floating anomaly score & Z-Score Threshold)	100	100	99.98	0.38
OCPAD (1-gram)	19.88	16.65	0.00	0.12
OCPAD (3-gram) (HTTP only)	29.08	23.31	8.85	0.23
PAYL	87.09	83.93	0.05	0.86
Wang et al. [110] (One-class SVM-based)	100	100	46.83	0.52
Wang et al. [110] (KNN-based)	36.41	100	0.03	0.41
Kitsune [78]	0	0	0.0004	0
Decanter [25] (HTTP Only)	68.13	7.62	0.02	0.15

This model achieved the highest F_2-score of 0.95. AE-OD with Z-score threshold, RNN-OD with floating anomaly score and Z-score threshold, and One-class SVM-based [110] may have 100% detection rate for both datasets. However, their FPR is also very high, making them infeasible to be implemented in real-life situations. Other methods either only worked well with one dataset (i.e., Decanter, and KNN-based [110]) or showed unsatisfactory performance (< 50% detection rate) in both datasets (i.e., OCPAD, the rest of RNN-OD, and Kitsune).

During the experiment with RNN-OD with binary anomaly score and T_{IQR} threshold method, we found that it performed best at analysing SMTP traffic with 100% detection rate and 1% false positive rate. It is because the byte sequences in SMTP messages in our dataset are more uniformly distributed than other protocols. Therefore when there is an exploit code, the sequence is highly unusual and easily detected by the method. On the other protocol, this method performed worst at analysing FTP attacks in BlattaSploit with a detection rate of 77.78%. We suspect that this result was caused by the FTP messages in the training data are relatively shorter than HTTP or SMTP messages. Therefore, the model has fewer samples for training. As for HTTP, this method has the highest rate of false positives among other protocols in the testing set.

To further analyse how some of our methods came up with a lower performance, we break down the result based on the protocol (i.e., HTTP, FTP, and SMTP) and the attack types (i.e., Backdoors, Exploits, Worms). AE-OD with T_{IQR} threshold seems to struggle with detecting low-rate attacks in UNSW-NB15 dataset. When we looked at the result in more detail, it is revealed that AE-OD with T_{IQR} threshold had difficulties in detecting HTTP-based exploits, only 30.5% of this type of attacks were identified. However, this method achieved 80% of detection rate for other attack types on HTTP. It even detected 100% of low-rate attacks on FTP and SMTP. RNN-OD with floating anomaly score and T_{IQR} threshold suffered a similar problem. In our further analysis, its low detection rate was caused by missing many HTTP-based exploits.

AE-OD with the Z-Score threshold has a reasonable detection rate, but the FPR is un-

acceptable in real-life situations. This result is caused by the method classifying all legitimate FTP traffic and 15.31% of legitimate HTTP traffic as malicious (see Appendix B. Both the number of layers and the number of neurons have no significant effect on the false positive rate. The results differ by around 0.5% or about 700 of 153,718 samples. Moreover, these parameters did not affect the detection rate at all; all protocol detection rates were 100%.

In RNN-OD, using the Z-score threshold method resulted in inferior performance regardless of our anomaly score calculation method. This poor performance can be seen in all protocols and attack types. Therefore, we argue that defining threshold with the Z-Score method is not suitable for RNN-OD. This method results in a higher threshold than it should be, causing malicious traffic to be deemed normal.

Suppose we look at the result from RNN-OD with floating anomaly score and T_{IQR} threshold method (see Appendix B, the detection rate on HTTP contributed badly to the overall performance. At best, the method can only detect 12.39% of HTTP-based attacks in UNSW-NB15 dataset and 57.87% of attacks in BlattaSploit. It might be caused by the way the anomaly score is calculated. Suppose the predicted output value of the RNN model is close to the actual value. In that case, although the prediction is said to be incorrect, the contribution to the overall anomaly score is smaller than the incorrect prediction with a big difference from the actual value. Therefore, causing the threshold to be less sensitive to malicious samples. Similar behaviours are also observed on SMTP and FTP albeit less damaging.

As previously mentioned, the one-class SVM model [110] with header-based features provided a higher detection rate than our best performing method, but it comes with 46.83% False Positive Rate (FPR), which would be unacceptable in a real-life environment. The KNN-based model [110] does capture 100% of malicious traffic in the BlattaSploit dataset, although it performed poorly on UNSW-NB15 dataset. This method could be considered the best performing method due to its ability to identify more malicious traffic in BlattaSploit, a dataset with more representative low-rate attacks with

lower false-positive, but it is worth noting that BlattaSploit was generated in a different environment from UNSW-NB15. They have different network topology and, thus, have really distinct feature values, such as the number of hops and inter-arrival packet time. Therefore, the method may have deemed all messages in BlattaSploit as malicious merely because of that difference. It is like training a method with data from one company and evaluating it with malicious data from another company. The model performance would be high, but it is misleading. Ultimately, the F_2-score shows that RNN-OD outperforms the KNN-based model [110]. It shows that RNN-OD works better at recognising attacks regardless of the dataset used to evaluate.

Other header-based NIDS, Kitsune [78], failed to detect any low-rate attacks at all. When we analysed the reconstruction errors generated by Kitsune, we found out that the reconstruction error of legitimate traffic has a massive range. Thus all low-rate attack anomaly scores fell into this range. Despite using an ensemble of Autoencoders, Kitsune with header-based features missed all low-rate attacks in our datasets. As our payload-based approaches, particularly the AE-OD that also utilises an Autoencoder, provide a better performance, this finding supports our argument that header-based features may not be suitable to capture the behaviour of low-rate attacks.

Our best performance model (RNN-OD) also provides improvements over other payload-based NIDS. Among others, OCPAD performed the worst. It detected less than 30% of low-rate attacks at best. It even performed worse than PAYL, the earlier approach. PAYL may show a decent result with a high detection rate and low false-positive rate. Its F_2-score is 0.86, the second highest after our best performance model. However, further analysis showed that it could only detect 42.93% of HTTP-based low-rate attacks. It struggled to detect exploits in HTTP traffic. We also show that Decanter [25] had trouble identifying HTTP low-rate attacks in UNSW-NB15 dataset, let alone attacks in BlattaSploit dataset. Furthermore, it only works on HTTP.

When conducting experiments, there were several issues we faced with the previous works. OCPAD [103] could not process SMTP traffic with 3-grams. It consumed too

much RAM. Even with 16 GB memory pre-allocated for the Java Virtual Machine (JVM), the JVM still threw an OutOfMemory exception. Kitsune [78] took four weeks to process our training and testing sets from UNSW-NB15 dataset and became slower over time.

4.5 Conclusions

In this chapter, we have presented two methods (i.e., RNN-OD and AE-OD) to detect low-rate attacks such as exploits, backdoors, and worms. Both have a slightly different approach to detect these attacks. To detect whether an application layer message is malicious, RNN-OD takes windowed byte subsequences from it and predicts the next byte for each subsequence. A malicious application layer message would cause RNN-OD to make incorrect predictions. When number of incorrect predictions surpasses a threshold value calculated in the training phase, the payload is flagged as malicious. On the other hand, AE-OD computes byte frequencies of the application layer message. The byte frequencies are then fed into an Autoencoder in which the output will be compared to the input byte frequencies. AE-OD deemed the application layer message malicious when the difference between its input and output surpasses the pre-calculated threshold in the training phase.

Our experiments show that the best performing model, a recurrent neural network with Long Short-Term Memory unit, combined with binary anomaly score and the statistical thresholding approach, was able to identify low-rate attacks in various application layer protocols (i.e., HTTP, FTP, and SMTP). In terms of detection rate, the proposed RNN-OD surpassed all previous works with an improvement of F_2-score at least 0.09. Therefore, none of the earlier works that have been included for comparison could match RNN-OD performance.

It has also been demonstrated that header-based NIDSs struggled at detecting low-rate attacks. Wang et al. [110]'s, which had been evaluated with KDD99 dataset and

showed a good result, was now evaluated with newer UNSW-NB15 dataset. Our experiments show that the models suffer from either high false-positive rate or low detection rate. The argument is also supported by Kitsune's failure to detect any low-rate attacks.

Chapter 5

Early prediction of low-rate attacks on network traffic with Recurrent Neural Networks

5.1 Introduction

An exploit is an example of low-rate attacks that takes advantage of the existence of bugs and vulnerabilities. They infiltrate the system by giving the system an input which triggers malicious behaviour. As time passes, the number of bugs and vulnerabilities increases, along with the number of exploits. In the first quarter of 2019, there were 400,000 new exploits [62], while more than 16 million exploits have been released in total. Only one of these is needed to infiltrate our system. More importantly, exploits are only a type of low-rate attacks. With the existence of other low-rate attacks, such as backdoors and worms, the threat to our system becomes more dangerous.

As demonstrated in Chapter 4, one way to detect low-rate attacks is to scan network traffic for their presence. However, the state of the art methods presented in Chapter 4, as well as the novel results we presented, all function by detecting the attack using a complete payload - i.e. after the exploit has completed. In this chapter we investigate whether an attack could be detected *before* it arrives at the vulnerable system. If this can be achieved, earlier action can be taken to minimise or nullify the damage. There

is also no need to dynamically analyse the attack on a clone server or *Virtual Machine* (VM) - as it is usually the case in host-based detection approaches, making this approach more time-efficient to block and provide rapid response to attacks. Therefore, detecting low-rate attacks in network traffic is a promising way to prevent low-rate attacks from infecting protected systems.

Detecting low-rate attacks on the wire, however, has challenges. Firstly, processing the vast amount of data without decreasing network throughput below acceptable levels; quality of service is still a priority. Secondly, there are various ways to encode the payloads [31], by modifying the payload to make it appear different, yet still, achieve the same goal. This technique makes it easy to evade any rule-based detection. Lastly, encrypted traffic is also a challenge; attackers may transmit an exploit with an encrypted protocol, e.g., HTTPS.

As discussed in Chapter 2, there are many ways to detect low-rate attacks in network traffic. Rule-based detection systems work by matching signatures of known attacks to the network traffic. Anything that matches the rule is deemed malicious. The most prevalent open-source intrusion detection system, Snort [95], has a rule that marks any traffic which contains byte 0x90 as shellcode-related traffic. This rule is based on the knowledge that most x86-based shellcodes are preceded by a sequence of *no operation* (NOP) instructions in which the bytes typically contain this value. However, this rule can easily be evaded by employing other NOP instructions, such as the "`0x41 0x49`" sequence. Apart from that, rule-based detection systems are susceptible to zero-day attacks for which no detection rule exists. Such systems are unable to detect these attacks until the rule database is updated with the new attack signature. Anomaly-based Network-based Intrusion Detection System (NIDS)s with Machine Learning (ML) algorithms offer a solution to this problem. And as demonstrated in Chapter 4, payload-based NIDSs are better at detecting low-rate attacks than header-based NIDSs.

One of the first leading research in payload analysis is PAYL [107]. It extracts 1-grams from all bytes of the payload as a representation of the network traffic. The model is

then trained over a set of those 1-grams. PAYL [107] measures the distance between the new incoming traffic with the model with a simplified Mahalanobis distance. Similar to PAYL, Oza et al. [82] extracts n-grams of HTTP traffic with various n values. They compared three different statistical models to detect anomalies/attacks. HMMPayl [21] is another work based on PAYL which uses Hidden Markov Models to detect anomalies. OCPAD [103] stores the n-grams of bytes in a probability tree and uses one-class Naïve Bayes classifier to detect malicious traffic. Golait and Hubballi [46] developed Probabilistic Counting Deterministic Timed Automata which inspects byte values of application layer messages to identify attacks on VOIP applications. Min et al. [77] extracts words from the application layer message and detect web-based attacks with a combination of Convolutional Neural Network (CNN) and Random Forest. In the previous chapter, we utilised Recurrent Neural Network (RNN)s with an unsupervised approach and windowed byte subsequences of payloads to improve on the detection rate of these previous methods by at least 12.04%. A common approach that is found on all of the works as mentioned above is they read all bytes in application layer messages and do not decide whether the payload is malicious or not until all bytes have been read. These messages can be lengthy and spread over multiple network packets. Reading the whole messages before making a decision may lead to a delay in detecting the attack and gives the attack time to execute before an alert is raised. To the best of our knowledge, this issue has never been addressed by the previous approaches. It then led us to a research question:

RQ3: *Can we predict the occurrence of low-rate attacks with fewer data and earlier in the attack, while still retaining a relationship between the sequence of packets?*

We, therefore, propose *Blatta*, an *early* low-rate attack prediction system which reads application layer messages and detects whether these messages are likely to be malicious by reading only the first few bytes. It is the first work to our knowledge that provides early prediction of malicious application layer messages, thus detecting a potential attack earlier than other state-of-the-art approaches, and enabling a form of an

early warning system.

In the previous chapter, RNN-OD takes windowed byte subsequences from application layer messages, uses each individual byte in a subsequence as an input for each time step, and outputs a byte value for each subsequence. The last two steps are repeated over the set of byte subsequences of an application layer message. For every subsequence, the predicted next-byte is compared to the actual one. RNN-OD raises an alert when the underlying RNN model makes many incorrect byte value predictions. The experiments with this approach showed an improvement over previous methods.

Inspired by the effectiveness of RNNs at detecting attacks in entire payloads in the previous chapter, the approach taken with Blatta in this chapter also utilises a recurrent neural network (RNN)-based model. There are three main differences between the RNN model used by RNN-OD (previous chapter) and Blatta (this chapter). The first difference is what is being processed by the RNN model in each time step calculation, or the input. Although in the preprocessing step RNN-OD obtains windowed byte subsequences from the application layer message, RNN-OD only takes a byte value as an input for each time step. On the other hand, the underlying RNN model in Blatta processes high-order n-grams ($n > 1$) in each time step. The second main difference between these two approaches is the output. The RNN model in RNN-OD outputs a byte value, representing the predicted next byte, as shown in Figure 4.4. There is an extra step in RNN-OD to decide whether the payload is malicious by counting the number of wrong byte predictions, rather than learning the sequences of malicious/non-malicious payloads. While Blatta directly outputs a decision on whether the payload is malicious.

To better illustrate these differences, we provide a sample of truncated Hypertext Transfer Protocol (HTTP) message in Table 5.1, which shows how each approach turns the message into features and what would be delivered as the output when the length of the subsequence is set to five. Each pair of square brackets represents the subsequence/n-grams which will be processed by the RNN model. Note that in this illustration, RNN-

OD only processes the first subsequence and further splits the subsequence into individual bytes. It then should output a space character as it is the expected byte value. This process is repeated for each subsequence in the square brackets, but for brevity, we only show the first calculation. In contrast to that, Blatta uses all the n-grams obtained from the payload as the input for the RNN model and should directly output a decision that this message is not malicious.

Table 5.1: An illustration of how RNN-OD and Blatta process the same HTTP message.

Sample HTTP message	GET / HTTP/1.0	
Method	**RNN-OD**	**Blatta**
Preprocessed subsequences/n-grams	[G, E, T, ' ', /]; [E, T, ' ', /, ' ']; [T, ' ', /, ' ', H]; [' ', /, ' ', H, T]; [/, ' ', H, T, T]; [' ', H, T, T, P]; [H, T, T, P, /]; [T, T, P, /, 1]; [T, P, /, 1, .]; [P, /, 1, ., 0]	['GET /', 'ET / ', 'T / H', ' / HT', '/ HTT', ' HTTP', 'HTTP/', 'TTP/1', 'TP/1.', 'P/1.0']
Input of the RNN model	'G', 'E', 'T', ' ', '/'	'GET /', 'ET / ', 'T / H', ' / HT', '/ HTT', ' HTTP', 'HTTP/', 'TTP/1', 'TP/1.', 'P/1.0'
Expected output of the RNN model	' ' (a space character)	Non-malicious payload

RNN-OD works by calculating an anomaly score over a full-length of an application layer message. If the number of bytes read from the message was reduced, the information passed to the model would also be reduced. It may cause the model to incorrectly classify the input because the number of error and the length of the message, two components for calculating the anomaly score, would be different. This difference would affect badly to the model accuracy if we made early prediction with RNN-OD.

Therefore, using RNN with the unsupervised approach may not be suitable for early prediction and requires further research to mitigate the potential performance drop. On the other hand, if we have labelled messages (i.e., malicious or legitimate) during the training, it is more straightforward for the RNN model to gradually learn whether a sequence of bytes is malicious in every time step. Therefore, it is possible to stop the calculation in an earlier time step while avoiding a significant accuracy drop. Blatta is built on that principle, which is why it uses a supervised approach. While changing the approach to supervised in this chapter may cause Blatta to lose some flexibility in detecting unseen attacks, we are more focused at looking for evidence of the ability to predict attacks earlier and need to use labelled data during the training phase.

There are several other payload-based NIDSs that incorporate RNNs in their approach [43, 92, 69, 52], yet none of them address the problem of early detection, all of them require the processing of the full-length payload before making a decision. Feng et al. [43] proposed a set of features from Modbus messages and an RNN-based model to detect attacks. Their approach requires reading the full-length Modbus messages before the feature set can be obtained. Moreover, their approach can only be used on Modbus messages as the features are not compatible with any TCP/IP-based protocols. [92, 69, 52] rely on individual bytes (1-grams) from the full-length payload as the features to their RNN model. Their RNN-based models take a sequence of bytes as input, process each item sequentially, and output the decision after it has finished processing the last item of the sequence. In the case of Blatta, the novel element is the ability to predict the exploit traffic early, by using the intermediate output of the RNN-based model, and not waiting for the full-length message to be processed. Our experiments show that this approach has little effect on accuracy, but enables us to make earlier network attack predictions while retaining high accuracy and a low false-positive rate. Therefore, Blatta is the first method that successfully addresses the problem of early detection of low-rate attacks.

To summarise, the key contribution of this chapter is:

C4: The first early low-rate attack prediction system on network traffic, which predicts malicious instances as they enter the protected network without analysing the whole application layer messages, enabling the administrator to react faster and possibly minimise the damage.

The rest of this chapter is structured as follows: Section 5.2 details the threat model which Blatta will face. How Blatta works is explained in Section 5.3. Then, Section 5.4 explains our extensive experimentation with Blatta. Finally, the chapter concludes in Section 5.5.

5.2 Threat Model

A threat model is a list of possible things that may affect protected systems. Having one means we can identify which part is the focus of our proposed approach, thus, in this case, we can potentially understand better what to look for to detect the malicious traffic and what the limitations are.

The proposed method focuses on detecting remote attacks by reading application layer messages from the unencrypted network traffic. However, the detection method of Blatta can be incorporated with application-layer firewalls, i.e., web application firewalls. Therefore, we can still detect the exploitation attempt before it reaches the protected application. In general, the type of attacks we consider here are:

(1) Low-rate attacks that send malicious scripting languages (e.g., PHP, Ruby, Python, or SQL), shellcode, or Bash scripts to maintain control to the server or gain access to it remotely. For example, apache_continuum_cmd_exec exploit with reverse shell payload which will force the targeted server to open a connection to the attacking computer and provide shell access to the attacker. By focusing on the connections directed to servers, we can safely assume JavaScript code in the application layer message could also be malicious since usually JavaScript code is sent from server to client, not the other way around.

(2) Low-rate attacks that utilise one of the text-based protocols over TCP, i.e., HTTP and FTP. Text-based protocols tend to be more well structured than binary-based ones. We also put more focus on HTTP and FTP as attacks on these protocols were harder to detect by our previous approach in Chapter 4 (i.e., RNN-OD and AE-OD).

5.3 Methodology

Extracting features from application layer messages is the first step toward an early prediction method. We could use variable values in the message (e.g., HTTP header values, FTP commands, SMTP header values), but it would require us to extract a different set of features from each application layer protocol. It is preferable to have a generic feature set which applies to various application layer protocols. Therefore, we propose the use of n-grams to model the application layer message.

Using 1-grams arguably overlooks information about the context of the payload string as a sequence of activities. One of the key benefits of an RNN model is the ability to learn sequences of activities. Therefore, we model the payload as sequences of high-order n-grams where $n > 1$ to capture more contextual information about the network payload. In doing so, Blatta takes the relation between consecutive subsequences of the application layer message into account. To test the effectiveness of higher order n-grams, we directly compare 1-grams to higher-order n-grams in our experiments. Moreover, while related work such as [59]'s, [41], and OCPAD [103] previously used high-order n-grams, they did not utilise a model capable of learning sequences of activities and thus not capable of making early-stage predictions within a sequence. Our novel RNN-based model will consider the long-term dependency of the sequence of n-grams.

A RNN takes a sequence of inputs and processes them sequentially in several time steps, enabling the model to learn the temporal behaviour of those inputs. In this case, the input to each time step is an n-gram, unlike earlier works [92, 69, 52] and our

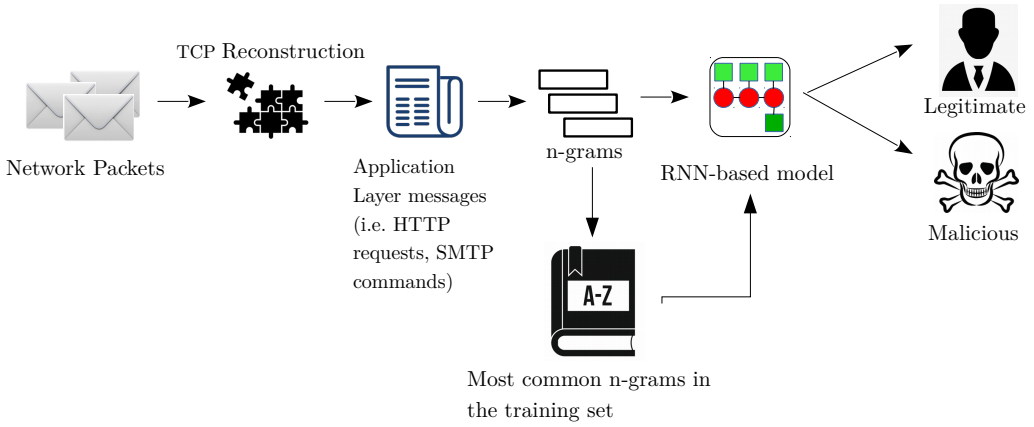

Figure 5.1: Architecture overview of Blatta. Application layer messages are extracted from captured traffic using tcpflow [44]. n-grams are obtained from those messages. They will then be used to build a dictionary of most common n-grams and train the RNN-based model (i.e., LSTM and GRU). The trained model output a prediction whether the traffic is malicious or benign..

previous method (i.e., RNN-OD) which also utilised an RNN model but took a byte value as the input for each RNN time step. Moreover, these works took the output from the last time step to make decision, while our novel approach produces classification outputs at intermediate intervals as the RNN model is already confident about the decision. We argue that this approach will enable the proposed system to predict whether a connection is malicious without reading the full length of application layer messages, therefore providing an early warning method.

In general, as shown in Figure 5.1, the training and detection process of Blatta are as follows:

Training phase. n-grams are extracted from application layer messages. l most common n-grams are stored in a dictionary. This dictionary is used to encode an n-gram to an integer. The integer encoded n-grams are then fed into an RNN model, training the model to classify whether the traffic is legitimate or malicious.

Detection phase. For each new incoming TCP connection directed to the server, we

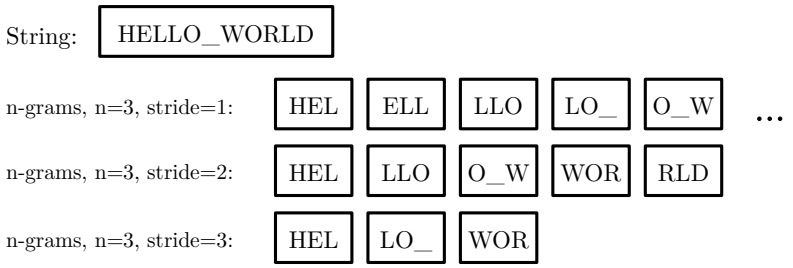

Figure 5.2: An example of n-grams of bytes taken with various stride values.

reconstruct the application layer message, obtain the the first few bytes of the message, and determine if the sequence belongs to malicious traffic.

5.3.1 Data Preprocessing

In well-structured documents such as text-based application layer protocols, byte sequences can be a distinguishing feature that makes each message in their respective class differ from each other. Blatta takes the byte sequence of application layer messages from network traffic. In the training phase, we reconstruct the application layer messages as the message may be split into multiple TCP segments and transmitted in an arbitrary order. For that purpose, we utilise `tcpflow` [44] to read PCAP files, reconstruct TCP segments, and obtain the application layer messages. While in the detection phase, only the first few bytes are obtained from the application layer message. The number of bytes taken from the message will be explored further in the experiments (see Section 5.4.3).

We then represent the byte sequence as a collection of n-grams taken with various *stride* value. An n-gram is a consecutive sequence of n items from a given sample, in this case, an n-gram is a consecutive series of bytes obtained from the application layer message. Stride is how many steps the sliding window takes when collecting n-grams. Figure 5.2 shows examples of various n-grams obtained with a different value of n and stride.

We define the input to the classifier to be a set of n-grams. Let $X = \{x_1, x_2, x_3, ..., x_k\}$ be the n-grams collected from an application layer message as the input to the RNN model, and k be the number of n-grams taken from each application layer message. The issue is, an RNN only accepts a sequence of numeric data or of vectors. Therefore, the input n-grams need to be encoded in a form that is accepted by the RNN model before processing further.

Each n-gram is categorical data. It means a value of 1 is not necessarily smaller than a value of 50. They are simply different. Encoding n-grams with one-hot encoding is not a viable solution as the resulting vector would be sparse and hard to model. Therefore, Blatta transforms the sequence of n-grams with *embedding* technique. Embedding is essentially a lookup table that maps an item to a dense vector with a fixed size dimension of $embedded\_dim$ [76]. In this case, each n-gram is transformed to a decimal vector with the size of $embedded\_dim$. Using pre-trained embedding vectors, e.g. GloVe [84], to initialise the embedding layer is common in natural language processing problems, but these pre-trained embedding vectors were generated from a corpus of words from Wikipedia, while our approach works with byte-level n-grams, and most of them do not exist in Wikipedia articles. Therefore, it is not possible to use the pre-trained embedding vectors. Instead, we initialise the embedding vectors with random values which will be updated by backpropagation during the training so that n-grams which usually appear together will have vectors that are close to each other.

It is worth noting that the number of n-grams collected raises exponentially as the n increases. If we considered all possible n-gram values, the model would overfit. Therefore, we limit the number of embedding vectors by building a dictionary of most-common n-grams in the training set. We define the dictionary size as l in which it contains l unique n-grams and a placeholder for other n-grams that do not exist in the dictionary. Thus the embedding vectors has $l + 1$ entries. However, we would like the size of each embedded vector to be less than $l + 1$. Let ε be the size of an embedded vector ($embedded\_dim$). If x_t represents an n-gram, the embedding layer transforms

X to $\hat{X}$. We denote $\hat{X} = \{\hat{x}_1, ..., \hat{x}_k\}$ where each $\hat{x}$ is a vector with the size of ε. The embedded vectors $\hat{X}$ are then passed to the recurrent layer.

5.3.2 Training an RNN-based classifier

Since the input to the classifier is sequential data, we opted to use a method that takes into account the sequential relationship of elements in the input vectors. Such methods capture the behaviour of a sequence better than processing those elements individually [47]. A *Recurrent Neural Networks* is an architecture of neural networks in which each layer takes time-series data, processes them in several time steps, and utilises the output of the previous time step in the current step calculation. We refer to these layers as recurrent layers. Each recurrent layer consists of recurrent units.

The vanilla RNN has a vanishing gradient problem, a situation where the recurrent model cannot be further trained because the value to update the weight is too small thus there would be no point of training the model further. Therefore, *Long Short-Term Memory* (LSTM) [54] and *Gated Recurrent Unit* (GRU) [35] are employed to avoid this situation. Both LSTM and GRU have cells/units that are connected recurrently to each other, replacing the usual recurrent units which existed in earlier RNNs. What makes their cells different is the existence of gates. These gates decide whether to pass certain information coming from the previous cell (i.e., input gate and forget gate in LSTM unit, update gate in GRU) or going to the next unit (i.e., output gate in LSTM). Since LSTM has more gates than GRU, it requires more calculations, thus computationally more expensive. Yet it is not conclusive whether one is better than the other [35]. Thus we use both types and compare the results. For brevity, we will refer to both types as recurrent layers and their cells as recurrent units.

The recurrent layer takes a vector $\hat{x}_t$ for each time step t. In each time step, the recurrent unit outputs hidden state h_t with a dimension of $|h_t|$. The hidden state is then passed to the next recurrent unit. The last hidden state h_k becomes the output of the

recurrent layer, which will be passed onto the next layer.

Once we obtain h_k, the output of the recurrent layer, the next step is to map the vector to benign or malicious class. Mapping h_k to those classes requires us to use linear transformation and softmax activation unit.

A linear transformation transforms h_k into a new vector L using Equation (5.1), where W is the trained weight and b is the trained bias. After that, we transform L to obtain $Y = \{y_i | \ 0 \ \leqslant \ i \ < \ 2 \}$, the log probabilities of the input file belonging to the classes with *LogSoftmax*, as described in Equation (5.2). The output of LogSoftmax is the index of an element that has the largest probability in Y. All these forward steps are depicted in Figure 5.3.

$$\mathrm{L} = W * h_k + b, \mathrm{L} = \{ \ l_i \mid 0 \ \leqslant \ i \ < \ 2 \ \} \tag{5.1}$$

$$\mathrm{Y} = \underset{0 \ \leqslant \ i \ < \ 2}{\arg\max} \left(\log \frac{\exp(l_i)}{\sum_{i=0}^{2} \exp(l_i)} \right) \tag{5.2}$$

In the training phase, after feeding a batch of training data to the model and obtaining the output, the next step is to evaluate the accuracy of our model. To measure our model's performance during the training stage, we need to calculate a loss value which represents how far our model's output is from the ground truth. Since this approach is a binary classification problem, we use *Negative Log Likelihood* [90] as the loss function. Then the losses are backpropagated to update weights, biases, and the embedding vectors.

5.3.3 Detecting Attacks

The process of detecting low-rate attacks is essentially similar to the training process. Application layer messages are extracted. n-grams are acquired from these messages

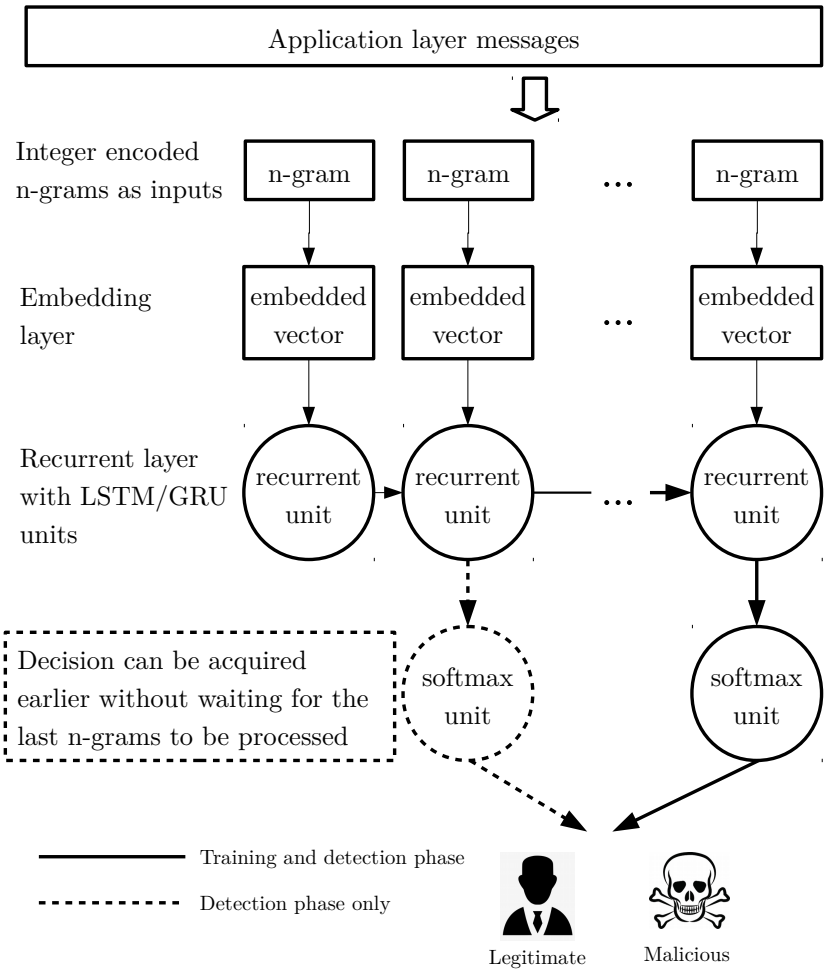

Figure 5.3: A detailed view of the classifier. n-grams are extracted from the input application layer message which are then used to train an RNN model to classify whether the connection is malicious or legitimate..

and encoded using the dictionary that was built during the training process. The encoded n-grams are then fed into the RNN model that will output probabilities of these messages being malicious. When the probability of an application layer message being malicious is higher than 0.5, the message is deemed malicious and an alert is raised.

The main difference in this process to the training stage is the time when Blatta stops processing inputs and makes the decision. Blatta takes the intermediate output of the

RNN model, hence requiring fewer inputs and disregarding the needs to wait for the full-length message to process. We will show in our experiment that the decision taken by using intermediate output and reading fewer bytes is close to reading the full-length message. Giving the proposed approach an ability of early prediction.

5.4 Experiments and Results

In this section, we evaluate Blatta and present evidence of its effectiveness in predicting low-rate attacks in network traffic. Blatta is implemented with Python 3.5.2 and PyTorch 0.2.0 library. All experiments were run on a PC with Core i7 @ 3.40GHz, 16 GB of RAM, NVIDIA GeForce GT 730, NVIDIA CUDA 9.0, and CUDNN 7.

The best practice for evaluating a machine learning approach is to have separate training and testing set. As the name implies, the training set is used to train the model, and the testing set is for evaluating the model's performance. We split BlattaSploit dataset in a 60:40 ratio for training and testing set as malicious samples. The division was carefully taken to include diverse types of exploit payloads. As samples of benign traffic to train the model, we obtained the same number of HTTP and FTP connections as the malicious samples from UNSW-JAN (see Section 4.4.2). Having a balanced set of both classes is essential in supervised learning.

We measure our model's performance by using samples of malicious and legitimate traffic. Malicious samples are obtained from 40% of BlattaSploit dataset (2,276 samples), exploits and worms samples in UNSW-FEB (see Section 4.4.2) set (10,855 samples). As for the legitimate traffic samples, we obtained the same number (10,855) of benign HTTP and FTP connections in UNSW-FEB. We used the UNSW dataset to compare our proposed approach performance with previous works.

In summary, the details of the training and testing sets used in the experiments are shown in Table 5.2.

Table 5.2: Numbers of benign and malicious samples used in the experiments.

Set	Obtained from	Num of samples	Class
Training Set	BlattaSploit	3,406	Malicious
	UNSW-JAN	3,406	Legitimate
Testing Set	BlattaSploit	2,276	Malicious
	UNSW-FEB	10,855	Legitimate
	UNSW-FEB	10,855	Malicious

We evaluated the classifier model by counting the number of true positive (TP), true negative (TN), false positive (FP), and false negative (FN). They are then used to calculate the detection rate (DR) and false-positive rate (FPR), which are metrics to measure our proposed system's performance.

DR measures the ability of the proposed system to detect malicious traffic correctly. The value of DR should be as close as possible to 100%. It would show how well our system able to detect low-rate attack traffic. The formula to calculate DR is shown in Equation 5.3. We denote TP as the number of identified malicious connections and FN as the number of undetected malicious connections in the testing set.

$$\mathrm{DR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{5.3}$$

FPR is the percentage of legitimate samples that are classified as malicious. We would like to have this metric to be as low as possible. High FPR means many false alarms are raised, rendering the system to be less trustworthy and useless. We calculate this metric using the Equation 5.4. We denote FP as the number of false positives detected and N as the number of benign samples.

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} \tag{5.4}$$

We now have two metrics to measure the performance. It is important to find the bal-

ance between improving DR and keeping FPR low. As we stated in the conclusion of Chapter 4, it depends on the cost of undetected attacks. Undetected low-rate attacks would be highly impactful as we would not have any idea the attack was happening. Thus no countermeasure could be done to prevent or mitigate the consequences. Therefore, we prioritise DR over FPR.

5.4.1 Data Analysis

Before discussing the results, it is preferable to analyse the data first to make sure that the results are valid, and the conclusion taken is on point. Blatta aims to detect attack traffic by reading the first few bytes of the application layer message. Therefore, it is crucial to know how many bytes are there in the application layer messages in our dataset. Hence, we can be sure that Blatta reads fewer bytes than the full length of the application layer message.

Table 5.3 shows the average length of application layer messages in our testing set. The median of message lengths in the benign samples is 474, lower than any other sets. Therefore, deciding after reading fewer bytes than at least that number implies our proposed method can predict malicious traffic earlier, thus providing improvements over previous works.

Table 5.3: Average message length of application layer messages in the testing set

Set	**Median of Message Lengths**
BlattaSploit	2285
UNSW-NB15 Legitimate samples	474
UNSW-NB15 Malicious samples	202

5.4.2 Comparison With Previous Works

We compare Blatta results with other related previous works. PAYL [107], OCPAD [103], Decanter [25], were included in the experiment as those are payload-based NIDS which are compared in the previous chapter and are unsupervised. Additionally, we also experimented with our unsupervised approaches, RNN-OD and AE-OD, described in Chapter 4. PAYL and OCPAD read an IP packet at a time, while Decanter, RNN-OD, and AE-OD reconstruct TCP segments and process the whole application layer message. None of them provides early detection, but to show that Blatta also offers improvements in detecting malicious traffic, we compare the detection and false-positive rates of those works with Blatta. In this experiment, Blatta still read the full-length application message, and the early prediction will be discussed further in Section 5.4.3.

We evaluated all methods with Exploits and Worms data in UNWS-NB15 as those match our threat model (see Section 5.2). Backdoors and Shellcode in UNSW-NB15, despite being a part of low-rate attacks, are excluded in this evaluation due to not having HTTP and File Transfer Protocol (FTP) traffic. It is also worth noting that although the training set for Blatta and other methods is slightly different, as Blatta is supervised and other methods are unsupervised, the testing set is the same. Blatta's training set is described in Table 5.2, while other methods did not include any traffic in BlattaSploit for training the model. The testing set in this experiment also did not use any traffic in BlattaSploit as a portion of it was used to train Blatta. We would like to investigate whether Blatta, which has not seen any malicious traffic in UNSW-NB15 dataset, can recognise low-rate attacks that do not exist during its training phase.

Blatta has a set of parameters that needs to be selected ahead of the training phase. These parameters are n, recurrent layer types, the number of recurrent layer(s), stride, dictionary size, and the embedding vector dimension. We will explore more about the effect of various values for these parameters later. As for this experiment, we set the parameters to the predefined default values (i.e., $n = 5$, $stride = 1$, dictionary size=2000, $embedding\_dim$=32, recurrent layer=LSTM, number of recurrent

Table 5.4: Comparison to previous works using the UNSW-NB15 dataset as the testing set.

Method	Detection Rate (%)		FPR (%)
	Exploits in UNSW-NB15	Worms in UNSW-NB15	
Blatta	99.04	100	1.93
PAYL	87.12	26.49	0.05
OCPAD	10.53	4.11	0.00
Decanter	67.93	90.14	0.03
RNN-OD	98.87	97.54	4.41
AE-OD	47.51	81.12	0.99

layers=1). The default values were selected based on a preliminary experiment, which had given the best result. Apart from the modifiable parameters, we set the optimiser to Stochastic Gradient Descent with learning rate 0.1 as using other optimiser did not necessarily increase or decrease the performance in our preliminary experiment. The number of epochs is fixed to five as the loss value did not decrease further, adding more training iteration did not give significant improvement.

The results of the experiments are shown in Table 5.4. In general, Blatta has the highest detection rate (99.04% and 100%) - albeit it also comes with the cost of increasing false positives (1.97%). Nevertheless, the increase in false positives does not negate the improvement in detection rate over previous methods. Moreover, Blatta detect the most low-rate attacks although it has never seen the attack in the training phase. Therefore, the supervised Blatta can perform on par with the unsupervised methods at detecting unknown attacks.

5.4.3 Early Prediction

In this section, we explore Blatta's capability to make an early prediction. Blatta makes an early prediction when it successfully identifies which class a TCP connection belongs to by reading bytes fewer than the full-length of its application message. There is always a possibility when Blatta ends up reading a full-length message because the message is shorter than the predefined limit. In that case, the prediction will not be deemed early. To see how many correct predictions are early, we define $early\_ratio$ as a ratio of correct predictions (i.e., true positive or true negative) that are made before reading the end of the message, TP_{early} and TN_{early} respectively, to the total correct prediction, TP and TN. Equation 5.5 defines how much malicious traffic is correctly detected before the method reads the last byte and Equation 5.6 defines how much of legitimate traffic is correctly detected before the method reads the last byte.

$$early\_ratio_p = \frac{\text{TP}_{early}}{\text{TP}} \tag{5.5}$$

$$early\_ratio_n = \frac{\text{TN}_{early}}{\text{TN}} \tag{5.6}$$

Before experimenting, we calculated the percentage of samples that are longer than some predefined byte limits. As shown in Table 5.5, more than 99% of legitimate samples are shorter than 500 bytes, and malicious samples in UNSW-NB15 tends to be shorter than the samples in BlattaSploit. This trend is also supported by the median of message lengths shown in Table 5.3.

During the experiment, we set Blatta to read full-length application layer messages first then we reduced the number of bytes read to 700, 600, 500, 400, 300, and 200 to see how the reduction would affect its performance at detecting low-rate attacks. Unlike the previous section, we used all samples in the testing set, including malicious traffic from BlattaSploit; hence the result is slightly different.

Table 5.5: The ratio of messages in each testing set that are greater than the byte limit..

	Set		
Byte limit	UNSW-NB15 legitimate samples	UNSW-NB15 malicious samples	BlattaSploit
200	88.97%	51.05%	99.43%
300	79.41%	26.80%	96.31%
400	70.81%	17.68%	89.50%
500	0.68%	13.78%	85.54%
600	0.66%	11.82%	77.33%
700	0.66%	10.91%	74.91%

We also explore how each parameter affects the detection and false-postiives rate of Blatta. In the beginning of the experiment, we set the parameters to their default values as described in Section 5.4.2. Then we tried different values for each parameter and analysed the effect.

In general, reading the full length of application layer messages mostly gives more than 99% detection rate with around 2.51% false-positive rate. This performance stays still with a minor variation when the length of input messages is reduced down to 500 bytes. When the length is limited to 400, the false positive rate spikes up for some configurations. Our hypothesis is benign samples have a relatively short length. Therefore, we will pay more attention to the results of reading 500 bytes or fewer and analyse each parameter individually.

n is the number of bytes (n-gram) taken in each time step. As shown in Table 5.6, we experimented with 1, 3, 5, 7, and 9-gram. For brevity, we omitted $n = 2, 4, 6, 8$ because the result difference is not significant. As predicted earlier, 1-gram were least effective. The detection rates were around 50%. As for the high-order n-grams, the detection rates are not much different, but the false-positive rates are. 5-gram and 7-gram provide better false-positive rates (2.51%) even when Blatta reads the first 400

bytes. 7-gram gives lower false-positive rate (8.92%) when reading first 300 bytes, yet it is still too high for real-life situations. Having a higher n means more information is considered in a time step. However, this may lead to overfitting instead of increasing performance.

As the default value of n is five, we experimented with a stride of one to five and presented the result in Table 5.7. It can be observed how the model would react depending on how much the n-grams overlapped. Non-overlapping n-grams provide lower false positives with around 1-3% decrease in detection rates. A value of two and three for the stride performs the worst. They missed quite a few malicious traffic.

The dictionary size plays a vital role in this experiment. Having too many n-grams leads to overfitting as the model would have to memorise too many of them that may barely occur. As shown in Table 5.8, we found that a dictionary size of 2000 has the highest detection rate and the lowest false positive rate. Reducing the size to 1000 has made the detection rate to drop for about 50%, even when the model read the full-length messages.

Blatta would have used a one-hot vector as big as the dictionary size for the input in a time step without the embedding layer. Therefore, the embedding dimension has the same effect as dictionary size. Having it too big leads to overfitting, too little could mean too much information is lost due to the compression. As presented in Table 5.9, our experiments show a dimension of 16, 32, or 64 give similar detection rates, differs less than 2%. An embedding dimension of 64 can have the least false positive when reading 300 bytes.

Changing the recurrent layer does not seem to have much difference. LSTM has a minor improvement over GRU as shown in Table 5.10. We argue that preferring one after the other would not make a significant improvement other than training speed. Adding more hidden layers does not improve the performance. On the other hand, it harms the detection speed, as shown in Table 5.12.

The experiment data also shows a substantial shift of performance between reading 300-400 bytes, particularly in term of the false positive rate. When we looked at the model's output for each time step, earlier time steps tend to classify everything as malicious when the model has limited information. As more data came in, the model became better at classifying. The change typically happened when reading more than 300 bytes. At that stage, the model typically has read the URL and the host header for HTTP and more than one commands for FTP. That information helps the model to be better at recognising legitimate traffic.

After analysing this set of experiments, we ran another experiment with a configuration based on the previously explained best-performing parameters. The configuration is n=5, stride=5, dictionary size=2000, embedding dimension=64, and a LSTM layer. The model then has a detection rate of 97.57% with 1.93% false positives by reading the first 400 bytes. By limiting the input to 400 bytes, Blatta has an $early\_ratio_n$ of 72.07%. As for the $early\_ratio_p$, we looked at the result of malicious samples in UNSW-NB15 and BlattaSploit separately. 88.35% correctly identified attacks in BlattaSploit (2,043 out of 2,276/89.76%) were made before the end of the message arrived, whilst only 17.18% correctly identified attacks (10,789 out of 10,855/99.39%) in UNSW-NB15 were early predicted. Considering that half of the number of malicious samples from UNSW-NB15 is shorter than 200 bytes, the result is not surprising. When we reduced the number of bytes read to 200, the overall detection dropped to 88.56%, and the false positive rate increased to 10.50%. Therefore, we suggest the configuration mentioned earlier, including the limit of 400 bytes, be used as a baseline for further research.

In this experiment, Blatta has been evaluated with malicious data from two datasets. Using the optimal configuration, the detection rates vary from 89.76% to 99.39% depending on the dataset used. We then expect Blatta to perform not too far from those values when it is evaluated on different datasets or real-world data. Considering we only have limited datasets with low-rate attacks, the robustness of Blatta on a broader

range of data remains to be seen, and further research is needed.

We conclude that Blatta can produce an accurate prediction before the end of the message for most samples. This result shows that Blatta retains high accuracy with an early ratio for legitimate traffic of 72.07% and 28.51% for malicious traffic. In the following section, we will show that the early prediction causes the detection speed to increase. For the subsequent experiment, we used the optimal set of parameters we mentioned earlier.

5.4.4 Detection Speed

In the IDS area, detection speed is another metric worth looked into, apart from accuracy-related metrics. Time is of the essence in detection, the earlier we detect malicious traffic, the faster we could react. However, detection speed is affected by many factors, such as the hardware or other applications/services running at the same time as the experiment. Therefore, in this section, we analyse the difference of execution time between reading the full and partial payload.

We first calculated the execution time of each record in the testing set, then divided the number of bytes processed by the execution time to obtain the detection speed in kilobytes/seconds (KBps). Eventually, the detection speed of all records was averaged. The result is shown in Table 5.12.

As shown in Table 5.12, reducing the processed bytes to 700, about half the size of an IP packet, increased the detection speed by approximately two times (from an average of 8.366 KBps to 16.486 KBps). Evidently, the trend keeps rising as the number of bytes reduced. If we take the number of bytes limit from the previous section, which is 400 bytes, Blatta can provide about three times increment in detection speed or 22.17 KBps on average. We are aware that this number seems small compared to the transmission speed of a link in the network which can reach 1 Gbps/128 MBps. However, we argued that there are other factors which limit our proposed method from perform-

ing faster, such as the hardware used in the experiment and the programming language used to implement the approach. Given the approach runs in a better environment, the detection speed will increase as well.

Table 5.6: Experiment results of using various n values and various length of input to the model. Bold values show the parameter value which gives the highest detection rate and lowest false positive rate.

		# of bytes							# of bytes						
Parameter		**All**	**700**	**600**	**500**	**400**	**300**	**200**	**All**	**700**	**600**	**500**	**400**	**300**	**200**
n	**1**	47.22	48.69	49.86	50.65	51.77	54.99	65.32	1.18	1.19	1.21	1.21	71.31	78.7	89.43
	3	99.87	99.51	99.77	99.10	99.59	98.93	91.07	2.51	2.51	2.51	2.51	72.61	10.29	20.51
	5	**99.87**	**99.55**	**99.78**	**99.57**	**99.29**	**98.91**	**88.75**	**2.51**	**2.51**	**2.51**	**2.51**	**2.51**	**72.60**	**11.08**
	7	99.86	99.47	99.59	99.37	99.19	98.53	97.08	2.51	2.51	2.51	2.51	2.51	8.92	80.92
	9	99.81	99.59	99.62	99.57	99.23	98.16	88.93	2.51	2.51	2.51	2.51	72.60	74.16	90.60
		Detection Rate							**False Positive Rate**						

Table 5.7: Experiment results of using various stride values and various length of input to the model. Bold values show the parameter value which gives the highest detection rate and lowest false positive rate.

		# of bytes							**# of bytes**						
Parameter		**All**	**700**	**600**	**500**	**400**	**300**	**200**	**All**	**700**	**600**	**500**	**400**	**300**	**200**
stride	**1**	99.87	99.55	99.78	99.57	99.29	98.91	88.75	2.51	2.51	2.51	2.51	2.51	72.60	11.08
	2	73.39	74.11	74.01	74.45	74.69	74.62	77.82	1.81	1.81	1.81	1.81	71.92	72.46	19.86
	3	82.51	82.54	83.07	83.12	83.25	83.50	85.75	1.50	1.49	1.50	1.51	71.62	75.47	89.63
	4	99.60	99.19	99.26	99.28	98.61	98.55	98.37	1.93	1.93	1.93	1.93	1.93	74.09	10.50
	5	**99.73**	**98.95**	**98.88**	**98.65**	**98.00**	**95.77**	**88.29**	**1.93**	**1.92**	**1.93**	**1.93**	**1.93**	**54.16**	**90.02**
		Detection Rate							**False Positive Rate**						

Table 5.8: Experiment results of using various dictionary size and various length of input to the model. Bold values show the parameter value which gives the highest detection rate and lowest false positive rate.

		# of bytes							# of bytes						
Parameter		**All**	**700**	**600**	**500**	**400**	**300**	**200**	**All**	**700**	**600**	**500**	**400**	**300**	**200**
dictionary	**1000**	47.78	49.50	50.36	50.79	51.80	54.83	54.68	1.21	1.21	1.22	1.22	71.33	79.47	89.42
size	**2000**	**99.87**	**99.55**	**99.78**	**99.57**	**99.29**	**98.91**	**88.75**	**2.51**	**2.51**	**2.51**	**2.51**	**2.51**	**72.60**	**11.08**
	5000	99.87	99.37	99.75	99.79	99.62	99.69	99.66	2.51	2.51	2.51	2.51	72.61	10.03	90.61
	10000	99.86	99.44	99.74	99.55	99.44	98.55	98.33	2.51	2.51	2.51	2.51	72.61	79.06	90.15
	20000	99.84	99.81	99.69	99.24	99.21	99.43	98.91	2.51	2.51	2.51	2.51	72.61	80.46	89.64
		Detection Rate							**False Positive Rate**						

Table 5.9: Experiment results of using various size of the embedding vector dimension and various length of input to the model. Bold values show the parameter value which gives the highest detection rate and lowest false positive rate.

		# of bytes							# of bytes						
Parameter		**All**	**700**	**600**	**500**	**400**	**300**	**200**	**All**	**700**	**600**	**500**	**400**	**300**	**200**
embedding	**16**	99.89	99.65	99.70	99.67	99.22	99.09	98.81	2.51	2.51	2.51	2.51	2.51	76.77	80.94
dimension	**32**	99.87	99.55	99.78	99.57	99.29	98.91	88.75	2.51	2.51	2.51	2.51	2.51	72.60	11.08
	64	**99.87**	**99.20**	**99.41**	**99.09**	**98.61**	**96.76**	**85.52**	**2.51**	**2.51**	**2.51**	**2.51**	**2.51**	**4.51**	**89.85**
	128	99.84	99.33	99.60	99.35	98.99	97.69	86.78	2.51	2.51	2.51	2.51	72.60	4.27	10.88
	256	99.88	99.76	99.80	99.22	99.38	98.64	90.34	2.51	2.51	2.51	2.51	72.60	80.79	90.60
		Detection Rate							**False Positive Rate**						

Table 5.10: Experiment results of using LSTM and GRU as the recurrent layer and various length of input to the model. Bold values show the parameter value which gives the highest detection rate and lowest false positive rate.

		# of bytes							# of bytes						
Parameter		**All**	**700**	**600**	**500**	**400**	**300**	**200**	**All**	**700**	**600**	**500**	**400**	**300**	**200**
recurrent layer	**LSTM**	**99.87**	**99.55**	**99.78**	**99.57**	**99.29**	**98.91**	**88.75**	**2.51**	**2.51**	**2.51**	**2.51**	**2.51**	**72.60**	**11.08**
	GRU	99.88	99.35	99.48	99.35	99.06	97.94	86.22	2.51	2.51	2.51	2.51	2.51	78.95	8.48
		Detection Rate							**False Positive Rate**						

Table 5.11: Experiment results of using number of recurrent layers and various length of input to the model. Bold values show the parameter value which gives the highest detection rate and lowest false positive rate.

		# of bytes							# of bytes						
Parameter		**All**	**700**	**600**	**500**	**400**	**300**	**200**	**All**	**700**	**600**	**500**	**400**	**300**	**200**
# of layers	**1**	**99.87**	**99.55**	**99.78**	**99.57**	**99.29**	**98.91**	**88.75**	**2.51**	**2.51**	**2.51**	**2.51**	**2.51**	**72.60**	**11.08**
	2	99.86	99.46	99.46	99.38	99.2	99.72	88.65	2.51	2.51	2.51	2.51	72.59	78.78	20.29
	3	99.84	99.38	99.68	99.1	99.18	98.16	87.35	2.51	2.51	2.51	2.51	2.51	74.94	10.83
		Detection Rate							**False Positive Rate**						

Table 5.12: The effect of reducing the number of bytes to the detection speed. The table shows the average (mean) detection speed in KBps with 95% confidence interval, calculated from multiple experiments. The detection speed increased significantly (about three times faster than reading the whole message), allowing early prediction of malicious traffic..

	# of LSTM Layers		
# of bytes	**1**	**2**	**3**
All	8.366 ± 0.238327	5.514 ± 0.004801	3.698 ± 0.011428
700	16.486 ± 0.022857	10.704 ± 0.022001	7.35 ± 0.044694
600	18.16 ± 0.020556	11.97 ± 0.024792	8.21 ± 0.049584
500	20.432 ± 0.02352	13.65 ± 0.036668	9.376 ± 0.061855
400	22.17 ± 0.032205	14.94 ± 0.037701	10.302 ± 0.065417
300	24.076 ± 0.022857	16.368 ± 0.036352	11.318 ± 0.083477
200	26.272 ± 0.030616	18.138 ± 0.020927	12.688 ± 0.063024

5.4.5 Visualisation

To investigate how Blatta has performed the detection, we took samples of both benign and malicious traffic and observed the input and output. We were particularly interested in the relation of n-grams that are not stored in the dictionary to the decision (unknown n-grams). Those n-grams either did not exist in the training set or were not common enough to be included in the dictionary.

On Figure 5.4, we visualise three samples of traffic taken from different sets, BlattaSploit and UNSW-NB15 datasets. The first part of each sample shows n-grams that did not exist in the dictionary. The yellow highlighted parts show those n-grams. The second part shows the output of the recurrent layer for each time step. The darker the red highlight, the closer the probability of the traffic being malicious to one in that time step.

As shown in Figure 5.4, malicious samples tend to have more unknown n-grams. It is evident that the existence of these unknown n-grams increases the probability of the traffic being malicious. As an example, the first five bytes of the five sample have around 0.5 probability of being malicious. And then the probability went up closer to one when an unknown n-grams are detected.

Similar behaviour also exists in the benign sample. The probability is quite low because there are many known n-grams. Despite the existence of unknown n-grams in the legitimate sample, the result shows that the traffic is legitimate. Furthermore, most of the time, the probability of the traffic being malicious is also below 0.5.

5.5 Conclusion and Future Work

This chapter presents Blatta, the first early prediction system for low-rate attacks which can detect malicious traffic by reading only 400 bytes from the application layer message. First, Blatta builds a dictionary containing most common n-grams in the training set. Then, it is trained over legitimate and malicious sequences of n-grams. Lastly, it continuously reads a small chunk of an application layer message and predicts whether the message will be a malicious one.

Decreasing the number of bytes taken from application layer messages only has a minor effect on Blatta's detection rate. Blatta does not need to read the whole application layer message like previously related works to detect low-rate attack traffic, creating a step change in the ability of system administrators to detect attacks early and to block them before the attack damages the system. Extensive evaluation of the new exploit traffic dataset has clearly demonstrated the effectiveness of Blatta. The result shows that Blatta only requires the first 400 bytes to achieve 97.57% detection rate with 1.93% false-positive rate with an early ratio for legitimate traffic of 72.07% and 28.51% for malicious traffic. In doing so, Blatta also provides a speed improvement by three-fold, in comparison to processing the full-length messages. This speed

Figure 5.4: Visualisation of unknown n-grams in the application layer messages and outputs of the recurrent layer for each time step. It shows how the proposed system observes and analyses the traffic. Yellow blocks show unknown n-grams. Red blocks show the probability of the traffic being malicious when reading an n-gram at that point..

improvement would help administrators react faster and minimise the damage to the system.

A model that can recognise malicious behaviour based on a sequence of messages exchanged between clients and a server would be an area which could be looked at in the future. Detecting attacks on encrypted traffic while retaining the early prediction capability could be a future research direction. It also remains to be seen whether the

approach can be modified to support training with unlabelled data.

Chapter 6

Conclusions

Low-rate attacks present threats to computer systems, particularly when they can be executed remotely by adversaries from anywhere in the world. Detecting such attacks on the network is challenging, yet may provide big rewards. Identifying the attack before it reaches the victim machines or services, could give administrators the ability to react faster and perhaps prevent the attack from achieving its full impact. Therefore, this thesis focuses on studying Network-based Intrusion Detection System (NIDS)s for identifying low-rate attacks.

6.1 Thesis Summary

In Chapter 2, we presented a literature review of existing works in NIDSs. In terms of detection methods, there are *signature-based* and *anomaly-based* NIDSs. Signature-based NIDSs, e.g., Snort [95] and Zeek [3], have a set of rules/signatures that represents what would be contained in malicious network traffic. If the exact signature is found in the traffic, then it is deemed as malicious. It is necessary to update the database of rules/signatures regularly, otherwise, this type of NIDSs typically will suffer from variation of known attacks or attacks which have never been seen, *zero-day* attacks. On the other hand, anomaly-based NIDSs learn the behaviour of network traffic and identify the malicious traffic, either by discovering the difference between legitimate and malicious traffic or by memorising the behaviour of legitimate traffic and looking

for anomalies. Several anomaly-based NIDSs incorporate Machine Learning (ML) algorithms as they are supposed to have more predictive power than statistical-based approaches [27].

In terms of the part of network traffic to analyse, there are *header-based* and *payload-based* NIDSs. A network packet consists of headers and payload. The headers contain information such as Internet Protocol (IP) addresses, ports, flags, packet lengths, and many others. There is also some information which is derived or aggregated from other information in the header part, e.g., inter-arrival time, average packet size, the total number of bytes in a Transmission Control Protocol (TCP) flow, and others. Header-based NIDSs process this kind of information to determine whether a packet or a flow is malicious, while payload-based NIDSs typically inspect the content or the application layer message of a TCP flow.

The literature review of various low-rate attack detection methods makes the following contribution:

C1: *A comparative study of state of the art supervised and unsupervised anomaly-based methods for detecting low-rate attacks with features obtained from the packet header and payload information to investigate the performance of existing methods and features when distinguishing low-rate attacks from legitimate traffic.*

In Table 2.2 and Table 2.3, it is shown that several previous works on NIDS were evaluated with old datasets, i.e., 1999 DARPA Intrusion Detection Evaluation Dataset (DARPA99) and KDD Cup 1999 Dataset (KDD99). These datasets are notorious for having multiple issues, particularly with not having representative samples of contemporary legitimate and malicious traffic [**?**]. Therefore, it makes the methods' ability in detecting contemporary low-rate attacks questionable. Since the release of DARPA99 and KDD99, other network traffic datasets have been published, such as UNB Intrusion Detection Evaluation Dataset 2012 (ISCX12), UNSW-NB15 and Gatech datasets. As the community has more options for evaluating their approach, it is deemed necessary to investigate whether these options are appropriate for contempor-

ary low-rate attack detection. It then led us to our first research question:

RQ1: *How well do existing validation datasets capture representative examples of contemporary low rate attacks?*

Chapter 3 aims to answer this question. In Chapter 3, we investigated the strengths and limitations of publicly available network traffic datasets, e.g., DARPA99, KDD99, NSL-KDD, ISCX12, UNSW-NB15, and Gatech datasets, and the result is shown in Table 3.2. Datasets that contain older traffic, such as DARPA99, KDD99, and NSL-KDD, tend to have a small proportion of low-rate attacks. More than 99% of malicious traffic in the datasets is either Distributed Denial of Services (DDoS) and Probe attacks. Gatech dataset may have more low-rate attacks than those datasets, but it still lacks contemporary attacks. Despite providing more contemporary traffic, more recent datasets still have several issues. ISCX12 lacks attack type information. Thus it is impossible to look for particular malicious traffic that contains a low-rate attack. UNSW-NB15 solved this issue by providing attack type labels for each flow. However, we found that some low-rate attack traffic in this dataset is indistinguishable from the legitimate traffic as it contains an arbitrary byte sequence and non-working exploits. Nevertheless, UNSW-NB15 contains the most recent traffic, in comparison to the other datasets, which is why it was still used to evaluate our proposed methods.

Another trait we have found with the existing datasets is that low-rate attack traffic may contain background packets as part of the protocol message. These background packets allow the victim to respond accordingly. However, none of the investigated datasets mention which part of the traffic contains the malicious sequence. This causes difficulty when validating whether detection by a payload-based NIDS is triggered by the malicious sequence or the background packets. Furthermore, it would not be possible for us to judge how early a payload-based NIDS can predict the occurrence of a low-rate attack without knowledge of the part of the payload that contains the malicious sequence.

To address this problem and complement UNSW-NB15 dataset, we generated an attack

dataset, *BlattaSploit*, that contains low-rate attacks, with 81.25% of them disclosed between 2010-2017. Each attack was executed with various combinations of attack payloads and encoders to make the generated traffic more diverse. The location of the malicious sequence is included in the dataset. To the best of our knowledge, BlattaSploit has the widest range of contemporary low-rate attacks, creating the following contribution:

C2: *A comparative study of network traffic datasets to better understand how representative they are to be used for evaluating NIDS and a dataset of low-rate attack traffic with state-of-the art attacks, various payloads, and encoders. Additionally, the dataset contains information about the location of the malicious traffic within the payload. The location is useful to analyse whether an early prediction method can detect the attack before it completes.*

Low-rate attacks are continually evolving. New attack vectors emerge regularly. Rule-based detection struggles to keep up with these changes as it will have to be updated with new attack signatures regularly. Researchers proposed anomaly-based NIDSs to address this issue. As we mentioned earlier, there are two approaches on how we get the representation of network traffic, from header information or the payload.

Older studies, which were evaluated with old datasets (i.e., KDD99 [113, 34, 60, 32, 42, 48, 96, 33, 110] and DARPA99 [107, 24, 108, 94, 21, 58, 82]), may struggle to differentiate more recent legitimate or malicious traffic. While more recent studies [111, 82, 51, 97, 109], despite showing improvements over the older ones, mostly focus on only one protocol (e.g., Hypertext Transfer Protocol (HTTP)). These two issues brought us to our second research question:

RQ2: *Given recent research has shown that previous work on low-rate attack detection across multiple protocols is dated and performance drops on contemporary low rate attacks, how do we improve the performance of low rate attack detection models to deal with evolving cyber attacks on a range of protocols that are increasingly causing damage to corporate networks?*

To address the problem of low-rate attack detection, several researchers opted to develop a detection method using a ML model. This model is commonly trained over a set of data to learn to distinguish legitimate and malicious traffic. The training can be supervised or unsupervised. Supervised learning relies on having samples for both classes. Therefore lack of malicious samples, which often the case in real-world situations, may lead to the difficulty of training the ML model. On the other hand, unsupervised models may be trained over samples of legitimate traffic only. They learn the behaviour of normal traffic and look for deviation to identify malicious traffic. Therefore, we proposed unsupervised Deep Learning (DL) models to detect low-rate attacks. DL-based models were chosen as they work better with abstract representation and a vast amount of data than conventional ML algorithms used by previous works (e.g., Logistic Regression [111], Support Vector Machine (SVM) [86], Hidden Markov Model (HMM) [21]).

We proposed the use of two DL architectures, Recurrent Neural Network (RNN)s and Autoencoders, to develop two separate detection models based on the principle of unsupervised learning. We refer to the RNN-based model as RNN-OD and the one with Autoencoders as AE-OD. These two methods have a slightly different approach to detect low-rate attacks, but both are trained with a set of legitimate traffic. RNN-OD works by obtaining windowed byte subsequences from an application layer message and predicting the next byte for each subsequence. A malicious application layer message would cause RNN-OD to make many incorrect guesses, surpassing the threshold value, which was calculated in the training phase. On the other hand, AE-OD computes byte frequencies of the application layer message. The byte frequencies are then fed into an Autoencoder model in which the output will be compared to the input byte frequencies. AE-OD deemed the application layer message malicious when the difference between its input and output surpasses the pre-calculated threshold in the training phase.

Our experiments explained in Section 4.4.4, used two metrics to measure the perform-

ance of our proposed methods and previous works, detection rate and false-positive rate. It is crucial to find the balance between these two metrics as increasing the detection rate could also come with the increase of the false-positive rate. This balance varies between cases and depends on the cost of undetected attacks. When the attack has severe effect to the system, which is the case for low-rate attack detection, it would be sensible to prioritise increasing the detection rate over the increase of false-positive rate since the cost of too many false-positive is more of an inconvenience [100]. Which is why F_2-score was also utilised as it prioritises detected attacks while still considering false positives.

It is shown in our experiments in Section 4.4.4 that the best performing model was able to identify low-rate attacks in various application layer protocols (i.e., HTTP, FTP, and SMTP). The best performing model is the recurrent neural network with Long Short-Term Memory (LSTM) unit (RNN-OD), combined with binary anomaly score and the statistical thresholding approach. In terms of detection rate, the proposed RNN-OD surpassed all previous works with an improvement of at least 12.04%. Albeit it comes with an increased false-positive rate of 3.52% over PAYL, the method with the second best detection rate. Therefore, none of the previous works that have been included for comparison could match RNN-OD performance.

It has also been demonstrated that header-based NIDSs struggled at detecting low-rate attacks. The failure of Kitsune [78] at detecting any low-rate attacks and the poor performance from Wang et al. [110]'s methods support the argument. Wang et al. [110]'s, which had been evaluated with KDD99 dataset and showed a good result, was now evaluated with newer UNSW-NB15 dataset. Our experiments show that the models suffer from either high false-positive rate or low detection rate. This result supports our earlier argument that works that methods which were evaluated with older datasets would struggle when facing more recent data. These results give support to the following contribution:

C3: *Unsupervised deep learning models to identify low rate attacks in network traffic,*

putting aside the requirement to provide malicious samples for the training data. The proposed approach offers an improvement in detection rate at least 12.04% from the previous works.

Due to the better performance of the payload-based NIDSs over header-based ones, as demonstrated by the experiments in Chapter 4, we looked for further improvements to payload-based NIDSs. Some application layer protocols may require long messages to be transmitted, such as HTTP and Simple Mail Transfer Protocol (SMTP). Processing the full-length messages would be time-consuming. The damage from the attack may have been done by the time the decision comes out. Therefore, we need an approach that can predict the occurrence of low-rate attacks from as little information as possible from the application layer message. To the best of our knowledge, none of the previous works has addressed this issue. It then led us to the third research question:

RQ3: *Can we predict the occurrence of low-rate attacks with fewer data and earlier in the attack, while still retaining a relationship between the sequence of packets?*

In Chapter 4, we proposed a novel low-rate attack detection method with an RNN. The effectiveness of RNNs at processing sequential data inspired us to apply a similar approach to the problem of early detection of attacks. More importantly, we can obtain an intermediary result from the RNN layers. It is not necessary to wait until the RNN finishes processing the last time step calculation. This trait makes RNN be capable of early prediction.

Blatta is an RNN-based low-rate attack detection system which is trained over samples of a full-length payload of legitimate and malicious traffic. However, in the detection phase, it only requires the first few bytes to decide whether an application layer message is malicious. Thus it provides the benefit of early prediction.

Blatta takes a sequence of high-order n-grams ($n > 1$) from the application layer message, use each n-gram as the input to the RNN model for each time step, and output a prediction whether the message is malicious. And as we mentioned earlier, Blatta

obtains the intermediate output of the RNN or the output of the RNN after several time steps, hence no need to wait for the full-length message to be processed.

To test the effectiveness of Blatta, we first evaluated it to detect low-rate attacks that were not the part of its training set and compared it with other unsupervised payload-based NIDSs, including our previous approaches, RNN-OD and AE-OD. The result shows that Blatta can still identify low-rate attacks with a high detection rate, even higher than any other methods. The false-positive rate of Blatta is also lower than RNN-OD. However, as part of its supervised nature, Blatta needs continual retraining with both new legitimate and malicious traffic so that it can still recognise the evolving attacks. While for unsupervised approaches this is not so much of a problem.

We then tested the early prediction capability of Blatta by analysing how many bytes it needs to provide detection and false-positive rate on par to the detection with full-length messages. The result shows that Blatta only requires the first 400 bytes to achieve 97.57% detection rate and 1.93% false positives rate with an early ratio for legitimate traffic of 72.07% and 28.51% for malicious traffic. In doing so, Blatta also provides a speed improvement by three-fold, in comparison to processing the full-length messages.

Blatta is the part of the following contribution:

C4: The first early low-rate attack prediction system on network traffic, which predicts malicious instances as they enter the protected network without analysing the whole application layer messages, enabling the administrator to react faster and possibly minimise the damage.

6.2 Discussion on Evasion Techniques, Adversarial Attacks, and Future Works

Our proposed approaches are not a silver bullet to tackle low-rate attacks. There are evasion techniques which could be employed by adversaries to evade the detection. These techniques may raise new questions and open possibilities for future works. Therefore, this section talks about such evasion techniques and discuss why our current methods have not covered them.

Since RNN-OD and AE-OD work by analysing restructured application layer messages, it is safe to disregard evasion techniques on transport or network layer level, e.g., IP fragmentation, TCP delayed sending, TCP segment fragmentation. They should be handled by the underlying tool that reconstructs TCP sessions. Those evasion techniques are usually handled by other defensive techniques, such as using Snort preprocessor. As they are well-known today, rules/signatures for such evasion techniques are already built into Snort. Such evasion techniques will also not work on Blatta as it only needs to read the first few bytes from the first packet after the TCP handshake.

Two possible evasion techniques are compression and/or encryption. Both compression and encryption change the byte values from the original and make the malicious code harder to detect by any payload-based detection. As an example, Metasploit has a collection of evasion techniques which include compression. The compression evasion technique, however, only works on HTTP and utilises `gzip`. This technique compresses HTTP responses but does not compress HTTP requests. All HTTP-based attacks in UNSW-NB15 and BlattaSploit have their malicious code in the request. Thus no compressed attack data is available to analyse the performance if the adversary uses compression. Apart from that, `gzip` compressed data could still be detected because it always starts with the magic number `1f 8b` and the decompression can be done in a streaming manner in which Blatta can do so. There is also no need to decompress the whole data since Blatta works well with partial input.

Encryption is possibly the biggest obstacle in payload-based NIDS: None of the previous payload-based NIDS in our literature (see Section 2.2.2) have addressed this challenge. There are other studies which deal with payload analysis in encrypted traffic [71, 15]. However, these studies focus on classifying which application generates the network traffic instead of detecting low-rate attacks. Thus they are not directly relevant to our research.

On its own, RNN-OD, AE-OD, and Blatta cannot detect attacks hiding in encrypted traffic. However, their model can be exported and incorporated with application-layer firewalls such as ShadowDaemon [7]. ShadowDaemon is commonly installed on a Web server and intercepts HTTP requests before being processed by the webserver software. It detects attacks based on its signature database. Since it is extensible and reads the same data as our methods (i.e., application-layer messages), it is possible to use one of our models to extend the capability of ShadowDaemon beyond rule-based detection. More importantly, this approach would enable our methods to deal with encrypted traffic, making it more applicable in real-life situations.

Another challenge which might be faced by Blatta is when attackers place the exploit code closer to the end of the application layer message. Hoping that in doing so, the attack would not be detected as Blatta reads merely the first few bytes. However, exploits with this kind of evasion technique would still be detected since this evasion technique needs padding to place the exploit code at the end of the message by putting dead code such as NOP. This is a common technique to evade a signature-based Intrusion Detection System (IDS) called *dead code insertion*. The dead code would still be recognised as a sign of malicious attempts as it is most likely to be a byte sequence which rarely exists in the legitimate traffic.

Any ML-based approaches are now susceptible to adversarial techniques. An adversarial ML works by crafting an input that can fool the detection method. The input can be supplied during the training phase. For example, suppose an adversary knows that an ML-based NIDS gathers legitimate data for the training model. In that case,

they may send malicious traffic so that it will be deemed legitimate. Therefore, the model will recognise such traffic as legitimate.

Another adversarial technique that can be used is crafting a malicious input that has similar features to legitimate traffic. For instance, if the adversary knew the byte frequency distribution of legitimate traffic in a network protected with AE-OD, they could craft a message containing malicious payload with the byte frequency of legitimate traffic. This evasion would be possible with the assumption that the adversary knows the legitimate byte frequency. This evasion technique opens up two directions of future research, how the adversary can make sure that the attack still works after modifying the message and how the defender can cope with this kind of evasion technique.

6.3 Conclusions

We predict low-rate attacks will still be recurring threats in the future based on the previous incidents. As new software occurring, there will always be vulnerabilities which could be exploited for the adversary benefit. Therefore, we have researched low-rate attack detection to reduce the risk of being attacked by such techniques. We started this thesis by conducting a systematic review that led us to analyse existing network traffic datasets, propose methods that improve the detection performance, and propose an early detection method that enables administrators to react faster and thus minimise the damage.

During the dataset analysis, we proposed six essential and one recommended metrics to justify whether a network traffic dataset is suitable for evaluating low-rate attack detection methods. This metrics may help companies measure how trustworthy the result of a ML-based detection method is based on the dataset used for its evaluation. For example, companies should be highly sceptical when they are presented with a great result when, in fact, the method was evaluated with DARPA99 dataset.

Based on the dataset comparison, we also decided to generate an attack traffic dataset,

BlattaSploit, that was used to complement UNSW-NB15 dataset to evaluate our subsequent detection methods. BlattaSploit contains low-rate attacks, 81.25% of which were disclosed between 2010-2017, making it an attack traffic dataset with the broadest range of new low-rate attacks. As we provide PCAP files in BlattaSploit, we hope the dataset can be widely used with any tools that extract information from raw network traffic. It will not limit researchers to develop new features, unlike preprocessed dataset such as KDD99 and NSL-KDD.

Chapter 4 presents our proposed detection methods (i.e. AE-OD and RNN-OD) that can be trained on unlabelled data. Our methods would be useful for companies or organisations that would like to train the detection method with their self-generated data. They do not need to spend time labelling the data. They could record the traffic for a period of time when an attack is unlikely to happen and use the captured traffic as the baseline for the normal model.

The experiment shows that the best performing model, a recurrent neural network with Long Short-Term Memory units, combined with binary anomaly score and the statistical thresholding approach, can identify low-rate attacks in various application layer protocols (i.e., HTTP, FTP, and SMTP). The proposed RNN-OD also surpassed all previous works with an improvement of F_2-score at least 0.09. Therefore, it gives more assurance to be used in a real-world situation than the previous related works.

The drawback of both AE-OD and RNN-OD is that they need to read the full-length application layer messages, potentially slowing down detection time. While they could still be used for real-time detection by scanning network traffic in real-time, if the traffic is big enough, it is suggested to use them as a forensic tool. For providing real-time detection, we proposed Blatta. Blatta can early predict the occurrence of low-rate attacks in an application layer message, providing faster detection time. In the experiment, Blatta was three times faster when reading fewer bytes while keeping the detection rate high. The detection speed improvement would help administrators be notified earlier and react faster, minimising damage to the system.

Nevertheless, the presented approaches are not perfect. None of the approaches has been evaluated against adversarial techniques. Suppose the adversarial technique is shown to be able to decrease the detection capability of the approaches. In that case, it is crucial for us to develop an improved method capable of handling adversarial inputs.

Adversarial machine learning is also another direction we can pursue. Existing adversarial techniques for NIDS typically only works on header-based NIDSs as their feature values are easier to change. It remains to be seen if a similar approach would work on the area of payload-based NIDSs. Modifying the attack may evade detection but not harm the system; the attack fails. Therefore, the adversarial technique should decrease the NIDS performance while keeping the attack works.

Appendices

Appendix A

List of low-rate attacks in BlattaSploit dataset

Table A.1: The list of attacks included in BlattaSploit dataset.

Module Name	**Vulnerability Name**	**Attack Type**	**Reference**	**Disclosure Year**
exploit/unix/webapp/generic_exec	Generic Web Application Unix Command Execution	Exploit		1993
exploit/unix/webapp/guestbook_ssi_exec	Matt Wright guestbook.pl Arbitrary Command Execution	Exploit	CVE-1999-1053	1999

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/unix/webapp/qtss_parse_xml_exec	QuickTime Streaming Server parse_xml.cgi Remote Execution	Exploit	CVE-2003-0050	2003
exploit/linux/http/linksys_apply_cgi	Linksys WRT54 Access Point apply.cgi Buffer Overflow	Exploit	CVE-2005-2799	2005
exploit/unix/webapp/barracuda_img_exec	Barracuda IMG.PL Remote Command Execution	Exploit	CVE-2005-2847	2005
exploit/unix/webapp/openview_connectednodes_exec	HP Openview connectedNodes.ovpl Remote Command Execution	Exploit	CVE-2005-2773	2005
exploit/unix/webapp/php_vbulletin_template	vBulletin misc.php Template Name Arbitrary Code Execution	Exploit	CVE-2005-0511	2005

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/unix/webapp/php_xmlrpc_eval	PHP XML-RPC Arbitrary Code Execution	Exploit	CVE-2005-1921	2005
exploit/unix/webapp/twiki_history	TWiki History TWikiUsers rev Parameter Command Execution	Exploit	CVE-2005-2877	2005
exploit/unix/webapp/wp_lastpost_exec	WordPress cache_lastpostdate Arbitrary Code Execution	Exploit	CVE-2005-2612	2005
exploit/linux/http/peercast_url	PeerCast URL Handling Buffer Overflow	Exploit	CVE-2006-1148	2006
exploit/linux/pop3/cyrus_pop3d_popsubfolders	Cyrus IMAPD pop3d popsubfolders USER Buffer Overflow	Exploit	CVE-2006-2502	2006
exploit/unix/webapp/pajax_remote_exec	PAJAX Remote Command Execution	Exploit	CVE-2006-1551	2006

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/unix/webapp/tikiwiki_jhot_exec	TikiWiki jhot Remote Command Execution	Exploit	CVE-2006-4602	2006
exploit/unix/webapp/tikiwiki_graph_formula_exec	TikiWiki tiki-graph_formula Remote PHP Code Execution	Exploit	CVE-2007-5423	2007
exploit/unix/webapp/coppermine_piceditor	Coppermine Photo Gallery picEditor.php Command Execution	Exploit	CVE-2008-0506	2008
exploit/unix/webapp/php_eval	Generic PHP Code Evaluation	Exploit		2008
exploit/unix/webapp/trixbox_langchoice	Trixbox langChoice PHP Local File Inclusion	Exploit	CVE-2008-6825	2008
exploit/linux/http/ddwrt_cgibin_exec	DD-WRT HTTP Daemon Arbitrary Command Execution	Exploit	CVE-2009-2765	2009

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/unix/http/contentkeeperweb_mimencode	ContentKeeper Web Remote Command Execution	Exploit	OSVDB-54552	2009
exploit/unix/webapp/dogfood_spell_exec	Dogfood CRM spell.php Remote Command Execution	Exploit	OSVDB-54707	2009
exploit/unix/webapp/nagios3_statuswml_ping	Nagios3 statuswml.cgi Ping Command Execution	Exploit	CVE-2009-2288	2009
exploit/unix/webapp/open_flash_chart_upload_exec	Open Flash Chart v2 Arbitrary File Upload	Backdoor	CVE-2009-4140	2009
exploit/unix/webapp/oscommerce_filemanager	osCommerce 2.2 Arbitrary PHP Code Execution	Exploit	OSVDB-60018	2009

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/unix/ftp/proftpd_133c_backdoor	ProFTPD-1.3.3c Backdoor Command Execution	Backdoor	OSVDB-69562	2010
exploit/unix/webapp/cakephp_cache_corruption	CakePHP Cache Corruption Code Execution	Exploit	CVE-2010-4335	2010
exploit/unix/webapp/mitel_awc_exec	Mitel Audio and Web Conferencing Command Injection	Exploit	OSVDB-69934	2010
exploit/unix/webapp/redmine_scm_exec	Redmine SCM Repository Arbitrary Command Execution	Exploit	CVE-2011-4929	2010
exploit/linux/http/vcms_upload	V-CMS PHP File Upload and Execute	Backdoor	CVE-2011-4828	2011

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/linux/http/webid_converter	WeBid converter.php Remote PHP Code Injection	Exploit	OSVDB-73609	2011
exploit/unix/http/ctek_skyrouter	CTEK SkyRouter 4200 and 4300 Command Execution	Exploit	CVE-2011-5010	2011
exploit/unix/webapp/mybb_backdoor	myBB 1.6.4 Backdoor Arbitrary Command Execution	Backdoor	OSVDB-76111	2011
exploit/linux/http/dlink_dir605l_captcha_bof	D-Link DIR-605L Captcha Handling Buffer Overflow	Exploit	OSVDB:86824	2012
exploit/linux/http/esva_exec	E-Mail Security Virtual Appliance learn-msg.cgi Command Injection	Exploit	OSVDB (85462)	2012

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/linux/http/hp_system_management	HP System Management Anonymous Access Code Execution	Exploit	OSVDB (91812)	2012
exploit/linux/http/symantec_web_gateway_exec	Symantec Web Gateway 5.0.2.8 ipchange.php Command Injection	Exploit	CVE-2012-0297	2012
exploit/linux/http/symantec_web_gateway_file_upload	Symantec Web Gateway 5.0.2.8 Arbitrary PHP File Upload Vulnerability	Backdoor	CVE-2012-0299	2012
exploit/linux/http/symantec_web_gateway_pbcontrol	Symantec Web Gateway 5.0.2.18 pbcontrol.php Command Injection	Exploit	CVE-2012-2953	2012
exploit/linux/http/wanem_exec	WAN Emulator v2.3 Command Execution	Exploit	OSVDB-85345	2012

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/linux/http/webcalendar_settings_exec	WebCalendar 1.2.4 Pre-Auth Remote Code Injection	Exploit	CVE-2012-1495	2012
exploit/linux/http/zen_load_balancer_exec	ZEN Load Balancer Filelog Command Execution	Exploit	OSVDB-85654	2012
exploit/linux/http/zenoss_showdaemonxmlconfig_exec	Zenoss 3 showDaemonXMLConfig Command Execution	Exploit	OSVDB-84408	2012
exploit/unix/http/freepbx_callmenum	FreePBX 2.10.0 / 2.9.0 callmenum Remote Code Execution	Exploit	CVE-2012-4869	2012
exploit/unix/webapp/basilic_diff_exec	Basilic 1.5.14 diff.php Arbitrary Command Execution	Exploit	CVE-2012-3399	2012

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/unix/webapp/egallery_upload_exec	EGallery PHP File Upload Vulnerability	Backdoor	OSVDB-83891	2012
exploit/unix/webapp/invision_pboard_unserialize_exec	Invision IP.Board unserialize() PHP Code Execution	Exploit	CVE-2012-5692	2012
exploit/unix/webapp/narcissus_backend_exec	Narcissus Image Configuration Passthru Vulnerability	Exploit	OSVDB-87410	2012
exploit/unix/webapp/projectpier_upload_exec	Project Pier Arbitrary File Upload Vulnerability	Backdoor	OSVDB-85881	2012
exploit/unix/webapp/spip_connect_exec	SPIP connect Parameter PHP Injection	Exploit	OSVDB-83543	2012
exploit/unix/webapp/wp_asset_manager_upload_exec	WordPress Asset-Manager PHP File Upload Vulnerability	Backdoor	OSVDB-82653	2012

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/unix/webapp/wp_foxypress_upload	WordPress Plugin Foxypress uploadify.php Arbitrary Code Execution	Backdoor	WPVDB-6231	2012
exploit/unix/webapp/wp_frontend_editor_file_upload	Wordpress Front-end Editor File Upload	Backdoor	WPVDB-7569	2012
exploit/unix/webapp/wp_property_upload_exec	WordPress WP-Property PHP File Upload Vulnerability	Backdoor	OSVDB-82656	2012
exploit/unix/webapp/wp_reflexgallery_file_upload	Wordpress Reflex Gallery Upload Vulnerability	Backdoor	CVE-2015-4133	2012
exploit/unix/webapp/xoda_file_upload	XODA 0.4.5 Arbitrary PHP File Upload Vulnerability	Backdoor	OSVDB-85117	2012

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/linux/http/cfme_manageiq_evm_upload_exec	Red Hat CloudForms Management Engine 5.1 agent/linuxpkgs Path Traversal	Exploit	CVE-2013-2068	2013
exploit/linux/http/dlink_command_php_exec_noauth	D-Link Devices Unauthenticated Remote Command Execution	Exploit	OSVDB:89861	2013
exploit/linux/http/dlink_diagnostic_exec_noauth	D-Link DIR-645 / DIR-815 diagnostic.php Command Execution	Exploit	CVE-2014-100005	2013
exploit/linux/http/dreambox_openpli_shell	OpenPLI Webif Arbitrary Command Execution	Exploit	OSVDB (90230)	2013
exploit/linux/http/netgear_readynas_exec	NETGEAR ReadyNAS Perl Code Evaluation	Exploit	CVE-2013-2751	2013

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/linux/http/pineapp_ldapsyncnow_exec	PineApp Mail-SeCure ldapsyncnow.php Arbitrary Command Execution	Exploit	OSVDB-95781	2013
exploit/linux/http/pineapp_livelog_exec	PineApp Mail-SeCure livelog.html Arbitrary Command Execution	Exploit	OSVDB-95779	2013
exploit/linux/http/raidsonic_nas_ib5220_exec_noauth	Raidsonic NAS Devices Unauthenticated Remote Command Execution	Exploit	EDB-24499	2013
exploit/linux/http/sophos_wpa_sblistpack_exec	Sophos Web Protection Appliance sblistpack Arbitrary Command Execution	Exploit	CVE-2013-4983	2013

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/linux/http/synology_dsm_sliceupload_exec_noauth	Synology DiskStation Manager SLICEUPLOAD Remote Command Execution	Backdoor	CVE-2013-6955	2013
exploit/linux/http/zabbix_sqli	Zabbix 2.0.8 SQL Injection and Remote Code Execution	Exploit	CVE-2013-5743	2013
exploit/linux/smtp/exim4_dovecot_exec	Exim and Dovecot Insecure Configuration Command Injection	Exploit	OSVDB-93004	2013
exploit/unix/webapp/carberp_backdoor_exec	Carberp Web Panel C2 Backdoor Remote PHP Code Execution	Backdoor		2013
exploit/unix/webapp/clipbucket_upload_exec	ClipBucket Remote Code Execution	Exploit	PACKETSTORM-123480	2013

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/unix/webapp/datalife_preview_exec	DataLife Engine preview.php PHP Code Injection	Exploit	CVE-2013-1412	2013
exploit/unix/webapp/flashchat_upload_exec	FlashChat Arbitrary File Upload	Backdoor	OSVDB-98233	2013
exploit/unix/webapp/graphite_pickle_exec	Graphite Web Unsafe Pickle Handling	Exploit	CVE-2013-5093	2013
exploit/unix/webapp/havalite_upload_exec	Havalite CMS Arbitary File Upload Vulnerability	Backdoor	OSVDB-94405	2013
exploit/unix/webapp/horde_unserialize_exec	Horde Framework Unserialize PHP Code Execution	Exploit	CVE-2014-1691	2013
exploit/unix/webapp/instantcms_exec	InstantCMS 1.6 Remote PHP Code Execution	Exploit	BID-60816	2013

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/unix/webapp/kimai_sqli	Kimai v0.9.2 'db_restore.php' SQL Injection	Exploit	OSVDB-93547	2013
exploit/unix/webapp/libretto_upload_exec	LibrettoCMS File Manager Arbitary File Upload Vulnerability	Backdoor	OSVDB-94391	2013
exploit/unix/webapp/openemr_upload_exec	OpenEMR PHP File Upload Vulnerability	Backdoor	CVE-2009-4140	2013
exploit/unix/webapp/php_charts_exec	PHP-Charts v1.0 PHP Code Execution Vulnerability	Exploit	OSVDB-89334	2013
exploit/unix/webapp/squash_yaml_exec	Squash YAML Code Execution	Exploit	CVE-2013-5036	2013

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/unix/webapp/vbulletin_vote_sqli_exec	vBulletin index.php/ajax/api/reputation/vote nodeid Parameter SQL Injection	Exploit	CVE-2013-3522	2013
exploit/unix/webapp/vicidial_manager_send_cmd_exec	VICIdial Manager Send OS Command Injection	Exploit	CVE-2013-4467	2013
exploit/unix/webapp/zimbra_lfi	Zimbra Collaboration Server LFI	Exploit	CVE-2013-7091	2013
exploit/linux/http/alienvault_sqli_exec	AlienVault OSSIM SQL Injection and Remote Code Execution	Exploit	CVE-2016-8581	2014
exploit/linux/http/fritzbox_echo_exec	Fritz!Box Webcm Unauthenticated Command Injection	Exploit	CVE-2014-9727	2014

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/linux/http/ipfire_bashbug_exec	IPFire Bash Environment Variable Injection (Shellshock)	Exploit	CVE-2014-6271	2014
exploit/linux/http/pandora_fms_exec	Pandora FMS Remote Code Execution	Exploit		2014
exploit/linux/http/pandora_fms_sqli	Pandora FMS Default Credential / SQLi Remote Code Execution	Exploit		2014
exploit/unix/http/twiki_debug_plugins	TWiki Debugenableplugins Remote Code Execution	Exploit	CVE-2014-7236	2014
exploit/unix/http/vmturbo_vmtadmin_exec_noauth	VMTurbo Operations Manager vmtadmin.cgi Remote Command Execution	Exploit	CVE-2014-5073	2014

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/unix/webapp/freepbx_config_exec	FreePBX config.php Remote Code Execution	Exploit	CVE-2014-1903	2014
exploit/unix/webapp/joomla_akeeba_unserialize	Joomla Akeeba Kickstart Unserialize Remote Code Execution	Exploit	CVE-2014-7228	2014
exploit/unix/webapp/projectsend_upload_exec	ProjectSend Arbitrary File Upload	Backdoor	EDB-35424	2014
exploit/unix/webapp/simple_e_document_upload_exec	Simple E-Document Arbitrary File Upload	Backdoor	EDB-31142	2014
exploit/unix/webapp/skybluecanvas_exec	SkyBlueCanvas CMS Remote Code Execution	Exploit	CVE-2014-1683	2014
exploit/unix/webapp/wp_creativecontactform_file_upload	Wordpress Creative Contact Form Upload Vulnerability	Backdoor	OSVDB-113669	2014

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/unix/webapp/wp_downloadmanager_upload	Wordpress Download Manager (download-manager) Unauthenticated File Upload	Backdoor	WPVDB-7706	2014
exploit/unix/webapp/wp_infusionsoft_upload	Wordpress InfusionSoft Upload Vulnerability	Backdoor	CVE-2014-6446	2014
exploit/unix/webapp/wp_symposium_shell_upload	WordPress WP Symposium 14.11 Shell Upload	Backdoor	OSVDB-116046	2014
exploit/linux/http/dlink_dcs931l_upload	D-Link DCS-931L File Upload	Backdoor	CVE-2015-2049	2015
exploit/linux/http/efw_chpasswd_exec	Endian Firewall Proxy Password Change Command Injection	Exploit	CVE-2015-5082	2015

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/linux/http/f5_icall_cmd	F5 iControl iCall::Script Root Command Execution	Exploit	CVE-2015-3628	2015
exploit/linux/http/pineapple_bypass_cmdinject	Hak5 WiFi Pineapple Preconfiguration Command Injection	Exploit	CVE-2015-4624	2015
exploit/unix/webapp/joomla_contenthistory_sqli_rce	Joomla Content History SQLi Remote Code Execution	Exploit	CVE-2015-7857	2015
exploit/unix/webapp/maarch_letterbox_file_upload	Maarch LetterBox Unrestricted File Upload	Backdoor	CVE-2015-1587	2015
exploit/unix/webapp/sixapart_movabletype_storable_exec	SixApart MovableType Storable Perl Code Execution	Exploit	CVE-2015-1592	2015
exploit/unix/webapp/wp_easycart_unrestricted_file_upload	WordPress WP EasyCart Unrestricted File Upload	Backdoor	CVE-2014-9308	2015

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/unix/webapp/wp_holding_pattern_file_upload	WordPress Holding Pattern Theme Arbitrary File Upload	Backdoor	CVE-2015-1172	2015
exploit/unix/webapp/wp_inboundio_marketing_file_upload	Wordpress InBoundio Marketing PHP Upload Vulnerability	Backdoor	WPVDB-7864	2015
exploit/unix/webapp/wp_nmediawebsite_file_upload	Wordpress N-Media Website Contact Form Upload Vulnerability	Backdoor	WPVDB-7896	2015
exploit/unix/webapp/wp_platform_exec	WordPress Platform Theme File Upload Vulnerability	Backdoor	WPVDB-7762	2015
exploit/unix/webapp/wp_worktheflow_upload	Wordpress Work The Flow Upload Vulnerability	Backdoor	WPVDB-7883	2015

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/unix/webapp/wp_wpshop_ecommerce_file_upload	WordPress WPshop eCommerce Arbitrary File Upload Vulnerability	Backdoor	WPVDB-7830	2015
exploit/linux/http/apache_continuum_cmd_exec	Apache Continuum Arbitrary Command Execution	Exploit	ExploitDB:39886	2016
exploit/linux/http/centreon_useralias_exec	Centreon Web Useralias Command Execution	Exploit	ExploitDB:39501	2016
exploit/linux/http/dlink_hnap_login_bof	Dlink DIR Routers Unauthenticated HNAP Login Stack Buffer Overflow	Exploit	CVE-2016-6563	2016
exploit/linux/http/ipfire_proxy_exec	IPFire proxy.cgi RCE	Exploit	ExploitDB:39765	2016
exploit/linux/http/kaltura_unserialize_rce	Kaltura Remote PHP Code Execution	Exploit	EDB-39563	2016

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/unix/webapp/drupal_coder_exec	Drupal CODER Module Remote Command Execution	Exploit		2016
exploit/unix/webapp/drupal_restws_exec	Drupal RESTWS Module Remote PHP Code Execution	Exploit		2016
exploit/unix/webapp/sugarcrm_rest_unserialize_exec	SugarCRM REST Unserialize PHP Code Execution	Exploit		2016
exploit/unix/webapp/tikiwiki_upload_exec	Tiki Wiki Unauthenticated File Upload Vulnerability	Backdoor		2016
exploit/linux/http/dnalims_admin_exec	dnaLIMS Admin Module Command Execution	Exploit	CVE-2017-6526	2017

Module Name	Vulnerability Name	Attack Type	Reference	Disclosure Year
exploit/linux/http/github_enterprise_secret	Github Enterprise Default Session Secret And Deserialization Vulnerability	Exploit	ExploitDB:41616	2017
exploit/linux/http/logsign_exec	Logsign Remote Command Injection	Exploit		2017
exploit/linux/http/trueonline_p660hn_v1_rce	TrueOnline / ZyXEL P660HN-T v1 Router Unauthenticated Command Injection	Exploit	CVE-2017-18368	2017
exploit/linux/http/wipg1000_cmd_injection	WePresent WiPG-1000 Command Injection	Exploit		2017
exploit/unix/webapp/wp_phpmailer_host_header	WordPress PHPMailer Host Header Command Injection	Exploit	CVE-2016-10033	2017

Appendix B

Experiment results of AE-OD and RNN-OD

Table B.1: The effect of various hidden layers configurations to the detection rate (DR) and false-positive rate (FPR) of AE-OD

Number of Neurons in Each Hidden Layer	**Threshold Method**	**DR**				**FPR**			
		HTTP	**FTP**	**SMTP**	**All**	**HTTP**	**FTP**	**SMTP**	**All**
200, 100	T_{IQR}	28.75	100.00	100.00	51.55	0.58	3.71	0.05	0.89
	Z-score	100.00	100.00	100.00	100.00	15.31	100.00	0.70	23.61
200	T_{IQR}	42.18	100.00	100.00	60.68	1.93	100.00	0.05	15.29
	Z-score	100.00	100.00	100.00	100.00	15.06	100.00	1.15	23.57
200, 100, 50	T_{IQR}	42.53	100.00	100.00	60.92	3.11	100.00	0.05	16.01
	Z-score	100.00	100.00	100.00	100.00	15.49	100.00	0.70	23.72

Table B.2: The effect of different length of subsequence and recurrent layer type (i.e., Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU)) to the detection rate (DR) and false-positive rate (FPR) of RNN-OD with binary anomaly score

Recurrent units	**Protocol(s)**	**Num of subsequence**	**Threshold method**					
			T_{IQR} **with Binary Anomaly Score**			**Z-Score with Binary Anomaly Score**		
			DR-UNSW	**DR-BS**	**FPR**	**DR-UNSW**	**DR-BS**	**FPR**
LSTM	**HTTP**	**3**	99.74	100.00	5.08	0.00	0.00	0.00
		5	99.92	100.00	5.09	0.00	0.00	0.00
		7	99.90	100.00	5.09	0.00	0.00	0.00
	FTP	**3**	90.50	77.78	1.54	0.00	0.00	0.00
		5	100.00	100.00	98.74	0.00	0.00	0.00
		7	96.51	60.00	2.27	0.00	0.00	0.00
	SMTP	**3**	100.00	100.00	1.00	0.00	0.00	0.00
		5	100.00	100.00	14.37	0.00	0.00	0.00
		7	100.00	100.00	8.18	0.00	0.00	0.00
	All	**3**	99.14	99.97	3.57	0.00	0.00	0.00
		5	99.95	100.00	20.58	0.00	0.00	0.00
		7	99.68	99.94	5.45	0.00	0.00	0.00

Recurrent units	Protocol(s)	Num of subsequence	Threshold method					
			T_{IQR} with Binary Anomaly Score			Z-Score with Binary Anomaly Score		
			DR-UNSW	DR-BS	FPR	DR-UNSW	DR-BS	FPR
GRU	HTTP	3	99.98	100.00	12.13	0.00	0.00	0.00
		5	99.74	100.00	5.07	0.00	0.00	0.00
		7	99.59	99.23	14.00	0.00	0.00	0.00
	FTP	3	97.24	90.91	1.77	0.00	0.00	0.00
		5	97.49	90.91	2.48	0.00	0.00	0.00
		7	92.61	75.00	1.20	0.00	0.00	0.00
	SMTP	3	100.00	100.00	0.06	0.00	0.00	0.00
		5	100.00	100.00	17.15	0.00	0.00	0.00
		7	100.00	96.10	3.30	0.00	0.00	0.00
	All	3	99.79	99.99	7.70	0.00	0.00	0.00
		5	99.64	99.99	7.68	0.00	0.00	0.00
		7	99.19	99.16	9.56	0.00	0.00	0.00

Table B.3: The effect of different length of subsequence and recurrent layer type (i.e., LSTM and GRU) to the detection rate (DR) and false-positive rate (FPR) of RNN-OD with floating anomaly score

Recurrent units	**Protocol(s)**	**Num of subsequence**	**Threshold method**					
			T_{IQR} **with Floating Anomaly Score**			**Z-Score with Floating Anomaly Score**		
			DR-UNSW	**DR-BS**	**FPR**	**DR-UNSW**	**DR-BS**	**FPR**
LSTM	**HTTP**	**3**	12.39	57.87	1.48	100.00	100.00	100.00
		5	2.47	18.84	1.43	100.00	100.00	100.00
		7	2.97	33.44	1.86	100.00	100.00	100.00
	FTP	**3**	68.10	44.44	1.49	100.00	100.00	99.92
		5	100.00	100.00	98.71	100.00	100.00	4.33
		7	78.00	70.00	2.30	100.00	100.00	99.59
	SMTP	**3**	84.56	77.92	0.02	100.00	100.00	100.00
		5	91.42	100.00	0.06	100.00	100.00	100.00
		7	83.33	100.00	0.06	100.00	100.00	100.00
	All	**3**	34.25	58.07	1.12	100.00	100.00	99.99
		5	31.49	19.86	14.82	100.00	100.00	86.50
		7	28.25	34.20	1.48	100.00	100.00	99.94

Recurrent units	Protocol(s)	Num of subsequence	Threshold method					
			T_{IQR} with Floating Anomaly Score			Z-Score with Floating Anomaly Score		
			DR-UNSW	DR-BS	FPR	DR-UNSW	DR-BS	FPR
GRU	HTTP	3	2.63	20.24	1.44	100.00	100.00	100.00
		5	12.30	26.21	1.45	100.00	100.00	100.00
		7	3.10	25.03	1.42	99.99	100.00	100.00
	FTP	3	8.04	0.00	95.64	100.00	100.00	99.92
		5	89.63	100.00	97.06	100.00	100.00	99.74
		7	32.14	62.50	2.25	100.00	100.00	99.60
	SMTP	3	17.18	5.19	0.00	100.00	100.00	100.00
		5	99.93	100.00	0.40	100.00	100.00	100.00
		7	87.49	100.00	0.06	100.00	100.00	100.00
	All	3	6.62	20.04	14.37	100.00	100.00	99.99
		5	39.55	27.13	14.68	100.00	100.00	99.96
		7	26.04	25.89	1.20	99.99	100.00	99.94

Bibliography

[1] Keras: The python deep learning library. URL https://keras.io/.

[2] Suricata open source ids/ips/nsm engine. URL https://suricata-ids.org/.

[3] The zeek network security monitor. URL https://www.zeek.org/index.html.

[4] Transmission control protocol, 1981. URL https://tools.ietf.org/html/rfc793.

[5] Snort product review, Nov 2009. URL https://www.scmagazine.com/review/snort/.

[6] Cve-2014-6271, 2014. URL https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271.

[7] Shadow daemon: a collection of tools to detect, record, and block attacks on web applications, 2015. URL https://shadowd.zecure.org/overview/introduction/.

[8] Cve-2017-0143, Sep 2016. URL https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-0143.

[9] CVE-2018-11932. Available from MITRE, CVE-ID CVE-2018-11932., Dec. 2 2018. URL https://nvd.nist.gov/vuln/detail/CVE-2018-11932.

[10] Wannacry ransomware attacks cost the nhs Â£92m. *Computer Fraud & Security*, 2018(11):1â3, 2018. doi: 10.1016/s1361-3723(18)30102-7.

[11] Cve-2019-1619, 2019. URL https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-1619.

[12] R. 7. Metasploit penetration testing framework, 2003. URL https://www.metasploit.com/.

[13] R. 7. Metasploitable, 2012. URL https://metasploit.help.rapid7.com/docs/metasploitable-2.

[14] L. Ablon and A. Bogart. *Zero days, thousands of nights: The life and times of zero-day vulnerabilities and their exploits*. Rand Corporation, 2017.

[15] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè. Mimetic: Mobile encrypted traffic classification using multimodal deep learning. *Computer Networks*, 165: 106944, 2019.

[16] M. E. Ahmed, S. Ullah, and H. Kim. Statistical application fingerprinting for ddos attack mitigation. *IEEE Transactions on Information Forensics and Security*, 14(6):1471–1484, 2018.

[17] C. C. f. A. I. D. Analysis. The caida anonymized internet traces dataset 2008 - ongoing, 2008. URL http://www.caida.org/data/passive/passive_dataset.xml.

[18] G. A. Anastassiou. Multivariate hyperbolic tangent neural network approximation. *Computers & Mathematics with Applications*, 61(4):809–821, 2011.

[19] B. Anderson and D. McGrew. Identifying encrypted malware traffic with contextual flow data. In *Proceedings of the 2016 ACM workshop on artificial intelligence and security*, pages 35–46. ACM, 2016.

[20] J. Andress. Network security. In *The Basics of Information Security*, pages 151–169. Elsevier, 2014. doi: 10.1016/b978-0-12-800744-0.00010-5. URL https://doi.org/10.1016/b978-0-12-800744-0.00010-5.

[21] D. Ariu, R. Tronci, and G. Giacinto. Hmmpayl: An intrusion detection system based on hidden markov models. *computers & security*, 30(4):221–241, 2011.

[22] K. Bartos, M. Sofka, and V. Franc. Optimized invariant representation of network traffic for detecting unseen malware variants. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 807–822, 2016.

[23] M. Boddy. *Exposed: Cyberattacks on Cloud Honeypots*. Apr 2019. URL `https://www.sophos.com/en-us/medialibrary/PDFs/Whitepaper/sophos-exposed-cyberattacks-on-cloud-honeypots-wp.pdf`.

[24] D. Bolzoni, S. Etalle, and P. Hartel. Poseidon: a 2-tier anomaly-based network intrusion detection system. In *Fourth IEEE International Workshop on Information Assurance (IWIA'06)*, pages 10–pp. IEEE, 2006.

[25] R. Bortolameotti, T. van Ede, M. Caselli, M. H. Everts, P. Hartel, R. Hofstede, W. Jonker, and A. Peter. Decanter: Detection of anomalous outbound http traffic by passive application fingerprinting. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, pages 373–386. ACM, 2017.

[26] S. T. Brugger and J. Chow. An assessment of the darpa ids evaluation dataset using snort. *UCDAVIS department of Computer Science*, 1(2007):22, 2007.

[27] D. Bzdok, N. Altman, and M. Krzywinski. Points of significance: statistics versus machine learning, 2018.

[28] R. S. M. Carrasco and M.-A. Sicilia. Unsupervised intrusion detection through skip-gram models of network behavior. *Computers & Security*, 78:187–197, 2018.

[29] X.-F. Chen and S.-Z. Yu. Cipa: A collaborative intrusion prevention architecture for programmable network and sdn. *Computers & Security*, 58:1–19, 2016.

[30] Z. Chen, C. K. Yeo, B. S. Lee, and C. T. Lau. Detection of network anomalies using improved-mspca with sketches. *Computers & Security*, 65:314–328, 2017.

[31] T.-H. Cheng, Y.-D. Lin, Y.-C. Lai, and P.-C. Lin. Evasion techniques: Sneaking through your intrusion detection/prevention systems. *IEEE Communications Surveys & Tutorials*, 14(4):1011–1020, 2012.

[32] Z. Chiba, N. Abghour, K. Moussaid, A. El Omri, and M. Rida. A novel architecture combined with optimal parameters for back propagation neural networks applied to anomaly network intrusion detection. *Computers & Security*, 75:36–58, 2018.

[33] Z. Chiba, N. Abghour, K. Moussaid, M. Rida, et al. Intelligent approach to build a deep neural network based ids for cloud environment using combination of machine learning algorithms. *Computers & Security*, 86:291–317, 2019.

[34] R. Chitrakar and C. Huang. Selection of candidate support vectors in incremental svm for network intrusion detection. *computers & security*, 45:231–241, 2014.

[35] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[36] J. A. Cid-Fuentes, C. Szabo, and K. Falkner. An adaptive framework for the detection of novel botnets. *Computers & Security*, 79:148–161, 2018.

[37] K. Cup. Dataset. *available at the following website http://kdd.ics.uci.edu/databases/kddcup99/kddcup99. html*, 1999.

[38] R. B. d'Agostino. An omnibus test of normality for moderate and large size samples. *Biometrika*, 58(2):341–348, 1971.

[39] J. J. Davis and A. J. Clark. Data preprocessing for anomaly based network intrusion detection: A review. *Computers & Security*, 30(6):353–375, 2011.

[40] D. M. Diez, C. D. Barr, and M. Cetinkaya-Rundel. *OpenIntro statistics*. CreateSpace, 2012.

[41] P. Duessel, C. Gehl, U. Flegel, S. Dietrich, and M. Meier. Detecting zero-day attacks using context-aware anomaly detection at the application-layer. *International Journal of Information Security*, 16(5):475–490, 2017.

[42] Z. Elkhadir and B. Mohammed. A cyber network attack detection based on gm median nearest neighbors lda. *Computers & Security*, 2019.

[43] C. Feng, T. Li, and D. Chana. Multi-level anomaly detection in industrial control systems via package signatures and lstm networks. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 261–272. IEEE, 2017.

[44] S. L. Garfinkel and M. Shick. Passive tcp reconstruction and forensic analysis with tcpflow. Technical report, NAVAL POSTGRADUATE SCHOOL MONTEREY CA, 2013.

[45] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[46] D. Golait and N. Hubballi. Detecting anomalous behavior in voip systems: A discrete event system modeling. *IEEE Transactions on Information Forensics and Security*, 12(3):730–745, 2016.

[47] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[48] J. Gu, L. Wang, H. Wang, and S. Wang. A novel approach to intrusion detection using svm ensemble with feature augmentation. *Computers & Security*, 2019.

[49] D. Guthrie, B. Allison, W. Liu, L. Guthrie, and Y. Wilks. A closer look at skip-gram modelling. In *LREC*, pages 1222–1225, 2006.

[50] D. Hadžiosmanović, L. Simionato, D. Bolzoni, E. Zambon, and S. Etalle. N-gram against the machine: On the feasibility of the n-gram network analysis for binary protocols. In *International Workshop on Recent Advances in Intrusion Detection*, pages 354–373. Springer, 2012.

[51] T. Hamed, R. Dara, and S. C. Kremer. Network intrusion detection system based on recursive feature addition and bigram technique. *Computers & Security*, 73: 137–155, 2018.

[52] Y. Hao, Y. Sheng, and J. Wang. Variant gated recurrent units with encoders to preprocess packets for payload-aware intrusion detection. *IEEE Access*, 7: 49985–49998, 2019.

[53] S. Hawkins, H. He, G. Williams, and R. Baxter. Outlier detection using replicator neural networks. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 170–180. Springer, 2002.

[54] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[55] M. Hubert and E. Vandervieren. An adjusted boxplot for skewed distributions. *Computational statistics & data analysis*, 52(12):5186–5201, 2008.

[56] B. Iglewicz and D. C. Hoaglin. *How to detect and handle outliers*, volume 16. Asq Press, 1993.

[57] A. Jamdagni, Z. Tan, P. Nanda, X. He, and R. Liu. Intrusion detection using geometrical structure. In *2009 Fourth International Conference on Frontier of Computer Science and Technology*, pages 327–333. IEEE, 2009.

[58] A. Jamdagni, Z. Tan, X. He, P. Nanda, and R. P. Liu. Repids: A multi tier real-time payload-based intrusion detection system. *Computer Networks*, 57(3): 811–824, 2013.

[59] X. Jin, B. Cui, D. Li, Z. Cheng, and C. Yin. An improved payload-based anomaly detector for web applications. *Journal of Network and Computer Applications*, 106:111–116, 2018.

[60] C. Khammassi and S. Krichen. A ga-lr wrapper approach for feature selection in network intrusion detection. *computers & security*, 70:255–277, 2017.

[61] S. Kottler, B. Anglin, N. Waisman, and K. Ballinger. February 28th ddos incident report, Mar 2018. URL `https://github.blog/2018-03-01-ddos-incident-report/`.

[62] M. Labs. *McAfee Labs Threat Report - August 2019*. Aug 2019. URL `https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-aug-2019.pdf`.

[63] P. Laskov, P. Düssel, C. Schäfer, and K. Rieck. Learning intrusion detection: supervised or unsupervised? In *International Conference on Image Analysis and Processing*, pages 50–57. Springer, 2005.

[64] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[65] S. Lee, J. Kim, S. Shin, P. Porras, and V. Yegneswaran. Athena: A framework for scalable anomaly detection in software-defined networks. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 249–260. IEEE, 2017.

[66] G. Lemaître, F. Nogueira, and C. K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017. URL `http://jmlr.org/papers/v18/16-365`.

[67] R. Lippmann, R. K. Cunningham, D. J. Fried, I. Graf, K. R. Kendall, S. E. Webster, and M. A. Zissman. Results of the darpa 1998 offline intrusion detection evaluation. In *Recent advances in intrusion detection*, volume 99, pages 829–835, 1999.

[68] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das. The 1999 darpa off-line intrusion detection evaluation. *Computer networks*, 34(4):579–595, 2000.

[69] H. Liu, B. Lang, M. Liu, and H. Yan. Cnn and rnn based payload classification methods for attack detection. *Knowledge-Based Systems*, 163:332–341, 2019.

[70] Q. LLC. Argus. URL `https://openargus.org/`.

[71] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing*, 24(3):1999–2012, 2020.

[72] M. V. Mahoney and P. K. Chan. An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 220–237. Springer, 2003.

[73] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, volume 89. Presses universitaires de Louvain, 2015.

[74] J. McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):262–294, 2000.

[75] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[76] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[77] E. Min, J. Long, Q. Liu, J. Cui, and W. Chen. Tr-ids: Anomaly-based intrusion detection through text-convolutional neural network and random forest. *Security and Communication Networks*, 2018, 2018.

[78] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection. *machine learning*, 5:2, 2018.

[79] Mitrecnd. Pynids, Jul 2014. URL `https://github.com/MITRECND/pynids`.

[80] N. Moustafa and J. Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *Military Communications and Information Systems Conference (MilCIS), 2015*, pages 1–6. IEEE, 2015.

[81] S. Muller, J. Lancrenon, C. Harpes, Y. Le Traon, S. Gombault, and J.-M. Bonnin. A training-resistant anomaly detection system. *Computers & Security*, 76:1–11, 2018.

[82] A. Oza, K. Ross, R. M. Low, and M. Stamp. Http attack detection using n-gram analysis. *Computers & Security*, 45:242–254, 2014.

[83] E. S. Pearson, R. B. D ââ'AGOSTINO, and K. O. Bowman. Tests for departure from normality: Comparison of powers. *Biometrika*, 64(2):231–246, 1977.

[84] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL `http://www.aclweb.org/anthology/D14-1162`.

[85] R. Perdisci, G. Gu, and W. Lee. Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 488–498. IEEE, 2006.

[86] R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee. Mcpad: A multiple classifier system for accurate payload-based anomaly detection. *Computer Networks*, 53(6):864–881, 2009.

[87] S. Ponomarev and T. Atkison. Industrial control system network intrusion detection by telemetry analysis. *IEEE Transactions on Dependable and Secure Computing*, 13(2):252–260, 2015.

[88] B. A. Pratomo, P. Burnap, and G. Theodorakopoulos. Unsupervised approach for detecting low rate attacks on network traffic with autoencoder. In *2018 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pages 1–8. IEEE, 2018.

[89] B. A. Pratomo, P. Burnap, and G. Theodorakopoulos. Blatta: early exploit detection on network traffic with recurrent neural networks. *Security and Communication Networks*, 2020.

[90] PyTorch. Negative log likelihood, 2016. URL `hhttps://pytorch.org/docs/stable/nn.html#nllloss`.

[91] T. Qin, Z. Liu, P. Wang, S. Li, X. Guan, and L. Gao. Symmetry degree measurement and its applications to anomaly detection. *IEEE Transactions on Information Forensics and Security*, 15:1040–1055, 2019.

[92] Z.-Q. Qin, X.-K. Ma, and Y.-J. Wang. Attentional payload anomaly detector for web applications. In *International Conference on Neural Information Processing*, pages 588–599. Springer, 2018.

[93] B. Rahbarinia, R. Perdisci, and M. Antonakakis. Segugio: Efficient behavior-based tracking of malware-control domains in large isp networks. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 403–414. IEEE, 2015.

[94] K. Rieck and P. Laskov. Language models for detection of unknown attacks in network traffic. *Journal in Computer Virology*, 2(4):243–256, 2007.

[95] M. Roesch et al. Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99, pages 229–238, 1999.

[96] B. Selvakumar and K. Muneeswaran. Firefly algorithm based feature selection for network intrusion detection. *Computers & Security*, 81:148–155, 2019.

[97] M. Shen, M. Wei, L. Zhu, and M. Wang. Classification of encrypted traffic with second-order markov chains and application attribute bigrams. *IEEE Transactions on Information Forensics and Security*, 12(8):1830–1843, 2017.

[98] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, 31(3):357–374, 2012.

[99] R. Shirey. Internet security glossary, version 2. RFC 4949, RFC Editor, August 2007. URL `https://tools.ietf.org/html/rfc4949`.

[100] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy*, pages 305–316. IEEE, 2010.

[101] W. Stallings and L. Brown. *Computer security*. Pearson Education (US), 2017.

[102] G. Stergiopoulos, A. Talavari, E. Bitsikas, and D. Gritzalis. Automatic detection of various malicious traffic using side channel features on tcp packets. In *European Symposium on Research in Computer Security*, pages 346–362. Springer, 2018.

[103] M. Swarnkar and N. Hubballi. Ocpad: One class naive bayes classifier for payload based anomaly detection. *Expert Systems with Applications*, 64:330–339, 2016.

[104] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–6. IEEE, 2009.

[105] S. R. A. I. Team, A. R. A. I. T. A. I. Team, and S. S. Response. Tortoiseshell group targets it providers in saudi arabia in probable supply chain attacks, Sep 2019. URL `https://www.symantec.com/blogs/threat-intelligence/tortoiseshell-apt-supply-chain`.

[106] N. Tripathi and N. Hubballi. Slow rate denial of service attacks against http/2 and detection. *Computers & security*, 72:255–272, 2018.

[107] K. Wang and S. J. Stolfo. Anomalous Payload-Based Network Intrusion Detection. In E. Jonsson, A. Valdes, and M. Almgren, editors, *Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004, Sophia Antipolis, France, September 15 - 17, 2004. Proceedings*, pages 203–222, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-30143-

1. doi: 10.1007/978-3-540-30143-1_11. URL http://dx.doi.org/10.1007/978-3-540-30143-1{_}11.

[108] K. Wang, J. J. Parekh, and S. J. Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *International Workshop on Recent Advances in Intrusion Detection*, pages 226–248. Springer, 2006.

[109] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti. Detecting android malware leveraging text semantics of network flows. *IEEE Transactions on Information Forensics and Security*, 13(5):1096–1109, 2017.

[110] W. Wang, J. Liu, G. Pitsilis, and X. Zhang. Abstracting massive data for lightweight intrusion detection in computer networks. *Information Sciences*, 433: 417–430, 2018.

[111] S. Whalen, N. Boggs, and S. J. Stolfo. Model aggregation for distributed content anomaly detection. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, pages 61–71. ACM, 2014.

[112] C. Wressnegger, G. Schwenk, D. Arp, and K. Rieck. A close look on n-grams in intrusion detection: anomaly detection vs. classification. In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, pages 67–76. ACM, 2013.

[113] J. Xiang, M. Westerlund, D. Sovilj, and G. Pulkkis. Using extreme learning machine for intrusion detection in a big data environment. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, pages 73–82. ACM, 2014.

[114] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[115] H. Zhang, D. D. Yao, N. Ramakrishnan, and Z. Zhang. Causality reasoning about network events for detecting stealthy malware activities. *computers & security*, 58:180–198, 2016.

[116] J. Zhang, R. Perdisci, W. Lee, X. Luo, and U. Sarfraz. Building a scalable system for stealthy p2p-botnet detection. *IEEE transactions on information forensics and security*, 9(1):27–38, 2013.