TECHNOLOGICAL UNIVERSITY OF THE SHANNON:
MIDLANDS MIDWEST

# Enhancing Cyber Attack Prevention and Detection using Application Process Tracing

by

Stephen Jacob

A thesis submitted for the degree of

Doctor of Philosophy in Engineering by Research

in the

Faculty of Engineering and Informatics
Department of Computer and Software Engineering

Supervisors:

Dr. Yuansong Qiao Dr. Brian Lee

August 2022

# Declaration of Authorship

I, STEPHEN JACOB, declare that this thesis titled, 'ENHANCING CYBER ATTACK PREVENTION AND DETECTION USING APPLICATION PROCESS TRACING' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

**Stephen Jacob**

Date:

# *Abstract*

Nowadays, software enterprises are being targeted by more advanced cyber-security threat models. Consequently, more sophisticated means of protecting software organisations are in high demand. Also, microservices are trending for being amongst the most popular software application design architecture. The aim of this thesis is to explore how application process tracing can be applied to enhance cyber-attack prevention and detection.

We propose two objectives for this research project. The first objective is to observe how the prediction of future events in an application thread can help identify potential targets and thus enable cyber-security personnel to take proactive defensive measures. This approach is valid for general business application processes. The second objective is to investigate how anomaly detection approaches can be applied to microservice application process tracing and detect seeded cyber-attacks.

One approach for addressing the first objective is to employ a machine learning model to learn general business application processes and functionality to provide a contextual oversight of the process application's infrastructure. This can be done by applying process mining to observe the execution paths of application processes. An alternative method is to employ a deep learning model to discover the contextual oversight of the application process. We trained a Long Short Term Memory (LSTM) model to learn the sequential dependencies for existing processes and subsequently made predictions in ongoing process instances with the aim of improving cyber situational awareness.

For addressing our second objective, we considered microservice application process tracing. The functionality of a microservices application can be monitored and logged using distributed tracing. Anomaly detection is defined as the discovery of irregular instances or patterns within a data series. To detect cyber-attacks in a microservices application, frequency distribution-based anomaly detection was performed to identify irregular microservice application activity within a synthetic data set of traces. This machine learning model was tested by simulating a brute force password guessing attack against the application.

To further address the second objective, the traffic of a microservices application can also be modelled using graph theory and anomaly detection techniques can also be applied to this model. In the last stage of our research, we trained a Diffusion Convolutional Recurrent Neural Network (DCRNN) using synthetic data sets of distributed traces to learn both the spatial and temporal dependencies of the data. Subsequently, we made predictions of microservice activity using traffic forecasting and applied threshold-based

anomaly detection to detect injected cyber-security attacks. The different cyber-attacks emulated in the testing data to evaluate this model include a brute force attack, a batch registration of bot accounts and a distributed denial of service attack.

# *Acknowledgements*

A Ph.D of four years is more than just an educational and working experience, but also a journey. It was an immensely interesting and eye-opening journey. I would like to thank the amazing, accomplished and friendly staff at TUS for the opportunity to work and learn alongside them. I wish to express my sincere appreciation and gratitude to my primary supervisor Dr. Brian Lee, Ph.D and the director of the Software Research Institute at the Athlone Campus of Technological University of the Shannon: Midlands Midwest for his guidance, support and feedback over the years. I would also like to thank my secondary supervisor Dr. Yuansong Qiao for his technical experience and his providing a suitable environment for conducting my Ph.D, and fellow researcher Yuhang Ye for providing insight and feedback for my work and collaboration in developing my published works. And to my fellow Post Graduate researchers, I thank you for making these four years at TUS a pleasant experience, even during the Covid-19 pandemic.

# Contents

# List of Figures

# List of Tables

# Abbreviations

| | |
|---|---|
| **AAE** | **A**dversarial **A**uto**E**ncoder |
| **AD** | **A**nomaly **D**etection |
| **AI** | **A**rtificial **I**ntelligence |
| **ANN** | **A**rtificial **N**eural **N**etwork |
| **ARP** | **A**ddress **R**esolution **P**rotocol |
| **BGP** | **B**order **G**ateway **P**rotocol |
| **BPTT** | **B**ack**P**ropagation **T**hrough **T**ime |
| **CAPTCHA** | **C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part |
| **CNN** | **C**onvolutional **N**eural **N**etwork |
| **CSIRT** | **C**omputer **S**ecurity **I**ncident **R**esponse **T**eam |
| **CSP** | **C**loud **S**ervice **P**roviders |
| **DCNN** | **D**iffusion **C**onvolutional **N**eural **N**etwork |
| **DCRNN** | **D**iffusion **C**onvolution **R**ecurrent **N**eural **N**etwork |
| **DDoS** | **D**istributed **D**enial of **S**ervice |
| **DL** | **D**eep **L**earning |
| **DNN** | **D**eep **N**eural **N**etwork |
| **DoS** | **D**enial of **S**ervice |
| **GAD** | **G**roup **A**nomaly **D**etection |
| **GCN** | **G**raph **C**onvolutional **N**etwork |
| **GRU** | **G**ated **R**ecurrent **U**nit |
| **ICMP** | **I**nternet **C**ontrol **M**essage **P**rotocol |
| **IDS** | **I**ntrusion **D**etection **S**ystem |
| **IM** | **I**nductive **M**iner |
| **LSTM** | **L**ong **S**hort **T**erm **M**emory |
| **MITM** | **M**an **I**n **T**he **M**iddle |

| | |
|---|---|
| **MSA** | **M**icro**S**ervices **A**rchitecture |
| **NLP** | **N**atural **L**anguage **P**rocessing |
| **NN** | **N**eural **N**etwork |
| **NoSQL** | **N**ot **O**nly **S**tructured **Q**uery **L**anguage |
| **OS** | **O**perating **S**ystem |
| **PaaS** | **P**latform **A**s **A** **S**ervice |
| **PAIS** | **P**rocess **A**ware **I**nformation **S**ystem |
| **PM** | **P**rocess **M**ining |
| **QoS** | **Q**uality of **S**ervice |
| **REST** | **RE**presentation **S**tate **T**ransfer |
| **RNN** | **R**ecurrent **N**eural **N**etwork |
| **RPC** | **R**emote **P**rocedure **C**all |
| **RSH** | **R**emote **SH**ell |
| **SaaS** | **S**ecurity **A**s **A** **S**ervice |
| **SDN** | **S**oftware **D**efined **N**etwork |
| **SGCN** | **S**patial **G**raph **C**onvolutional **N**etwork |
| **SNMP** | **S**imple **N**etwork **M**anagement **P**rotocol |
| **SOM** | **S**elf-**O**rganized **M**aps |
| **TCP** | **T**ransport **C**ontrol **P**rotocol |
| **VAE** | **V**ariational **A**uto**E**ncoder |
| **VPN** | **V**irtual **P**rivate **N**etwork |
| **WF-Nets** | **W**ork**F**low Nets |
| **XSS** | **C**ross-**S**ite **S**cripting |

# Chapter 1

# Introduction

## 1.1 Content & Motivation

Due to the ongoing advancement of software computing, new cyber-security threats are being developed every day. Consequently, software enterprises are constantly being targeted and attacked by more advanced cyber-attacks. Therefore, more sophisticated means of protecting software organisations are in high demand. One countermeasure Computer Security Incident Response Teams (CSIRT) could implement for their software enterprises is to employ a defined model representing their system application which provides a contextual oversight of their general business process application infrastructure. Using this specified model, cyber-security personnel could detect impending cyber-attacks and identify the intended attack targets of their system application. This allows cyber defenders to enable proactive defensive measures and efficiently allocate appropriate personnel and resources to address the cyber-threat.

In a system application, the aims, services, and infrastructure of the application can be modelled to provide an illustrated overview of the entire organisation's application. One method to do this is the application of process mining techniques which takes an application's functionality in the form of logged data as input and returns a process model. This model can be used to observe and analyse the execution paths of the application. An alternative to traditional process mining is to train a Deep Neural Network (DNN) to learn the sequential process behaviour of the data and make predictions about future events. For the first contribution of this project, it is proposed to use process mining and

deep learning methods to improve cyber situational awareness by providing a contextual oversight of a general process application. This provides cyber-security personnel with advance warning of impending threats to their application's critical assets which supports the prevention of cyber-attacks.

Nowadays, the microservices architecture (MSA) has become the designated software application design due to its loosely coupled, service-oriented architecture. In microservices, an application is decomposed into multiple independent components, each of which is responsible for a single business function of the application. This polylithic design facilitates faster and more efficient software application development in contrast to the monolithic application architecture which must be developed and scaled in its entirety. Microservices have been adopted as the primary software application architecture for many popular software organisations including Netflix, eBay, and Amazon.

Anomaly detection is defined as the discovery of unusual instances or patterns within a data series. These anomalies either do not conform to the behaviour of the majority of the data or appear at a greater or lesser frequency than regular data instances. In an operational microservices application, the overall functionality can be monitored using distributed tracing. Distributed traces are logged records of microservice calls to the application at run-time. A frequency distribution of traces can be generated from the data and anomalous traffic caused by an attack could be detected as an anomaly if it differs from the normal distribution. For the second contribution, an open-source microservice application called DeathStarBench was configured with a distributed tracing tool and executed to generate and log synthetic microservice traffic, and employed a frequency distribution-based anomaly detection technique to detect a seeded cyber-attack targeting the microservices application. This supports the enhancement of cyber-attack detection.

For the third contribution of this research, the previous work for using distributed tracing and anomaly detection to detect seeded cyber-attacks targeting the microservices application, DeathStarBench was extended. For this work, a Deep Learning (DL) model called the Diffusion Convolutional Recurrent Neural Network (DCRNN) was trained to discover and learn the spatial and temporal dependencies of the microservice tracing data. Subsequently, the trained DCRNN model used traffic forecasting to predict future microservice activity. Subsequently, threshold-based anomaly detection was applied

against the predicted microservice data to detect irregular microservice functionality that indicates seeded cyber-security attacks. The cyber-attacks injected into the microservice traffic for this experiment include a brute force password guessing attack, a batch registration of bot accounts, and a Distributed Denial-of-Service attack.

For our overall research thesis, we propose two research objectives stated as follows: 1) to investigate how process flow prediction in application process threads can improve cyber situational awareness and 2) to explore how anomaly detection approaches can be applied to microservice process traces to detect cyber-attacks.

The findings of this research project were published in articles for international conferences and journal with a common theme of Artificial Intelligence (AI) and Cyber-Security.

## 1.2 Research Questions & Contributions

1. *"Can process mining and deep learning models predict next steps in an application process?"*

   Due to the numerous cyber-security intrusions that target a single software organisation daily, an IDS generates hundreds of logged cyber-attack alerts in their host computer systems. A problem with these numerous alerts is that cyber-security personnel will find it difficult to distinguish the more annoying and low-priority intrusions from the more severe attacks that signal a more serious threat to their software enterprises. One approach to this problem is to discover a contextual oversight of general application processes to predict future events in ongoing application process flow threads which improves cyber situational awareness. This supports personnel in identifying critical assets and intended targets of their software application. They are given advance warning of impending cyber-attacks, which supports cyber-security attack prevention. An attempted cyber-attack logged by an IDS can be represented as an aggregation of related cyber-attack alerts. This logged data set can be used as input data for a process mining tool. Also, a DL network model can be used to learn the process-oriented application workflow of the data and discover the contextual oversight of the process application. The main contribution for the first part of this thesis is a Long Short Term Memory

(LSTM) model which is trained to discover and learn the sequential dependencies within a data set and subsequently predict future events in ongoing process instances. This LSTM model addresses the first objective of this research project to investigate how process flow prediction can be used to improve cyber situational awareness which supports cyber attack prevention.

2. *"Can distributed tracing and anomaly detection detect cyber-security attacks in a microservices application?"*

   The overall functionality of a microservices-based application can be logged and monitored using distributed tracing which monitors and logs detailed information about a user's HTTP API call to a microservice. The distributed traces returned record the name of the microservice called, the time of their execution and the duration of the microservice call. For this part of the thesis, initial work is performed to apply anomaly detection to determine the presence of irregular microservice calls caused by a cyber-security attack. A frequency distribution of various microservice functionality can be extracted from normal data and anomalous activity detected if there is a significant difference in frequency of traces from the base distribution of normal data. This frequency distribution-based anomaly detection model for microservices is the second contribution for the research project and addresses the second research objective to explore how anomaly detection can be applied to detect cyber-attacks in microservice process traces. This part of the thesis only covers that anomaly detection can be applied based on the sequential execution paths of microservices propagating throughout an application. To learn both the sequential and temporal dependencies within the traces, it is proposed to represent microservice calls as active traffic modelled along a temporal series using graph theory. This leads to the following research question and the final part of the thesis.

3. *"Can graph convolutional neural networks and traffic forecasting detect cyber security attacks in microservices applications?"*

   For the final part of the research project, the work from the previous research question regarding the application of anomaly detection to detect anomalous microservice functionality caused by cyber-attacks is expanded upon. The ideal approach would be to train a DL model to learn the normal microservices application behaviour over a time series, perform traffic forecasting using the DL model to predict

future microservice activity. Due to the complexity of a microservices application architecture and its reliance on collaborative messaging between the individual microservice components, it is difficult to monitor and analyse the application's functionality. This makes it necessary to model the application topology as a graph where the microservice instances are represented as nodes and the calls sent to these microservices are represented as the connections. These API RPCs sent by users to the microservices over a time series can be modelled as a diffusion process originating from one microservice node to its neighbours. A graph convolutional network model called the DCRNN can be trained using the distributed tracing data and capture the spatial and temporal dependencies of the diffusion-based representation of the microservice application traffic. Finally, threshold-based anomaly detection can be used to detect cyber-security attacks injected into the microservices traffic. This DCRNN model and anomaly detection approach together are the primary contribution for the third research question, and also addresses the second research objective to explore the application of anomaly detection to detect cyber-attacks in microservice process traces.

## 1.3 Publications

Here, the research articles published are presented which outline the main contributions to the proposed objectives and research questions for this thesis.

1. Using Recurrent Neural Networks to Predict Future Events in a Case with Application to Cyber Security

Stephen Jacob, Yuansong Qiao, Paul Jacob, and Brian Lee. Using recurrent neural networks to predict future events in a case with application to cyber security. In BUSTECH, pages 13–19, 2020.

2. Detecting Cyber Security Attacks against a Microservices Application using Distributed Tracing

Stephen Jacob, Yuansong Qiao, and Brian Lee. Detecting cyber security attacks against a microservices application using distributed tracing. In ICISSP, pages 588–595, 2021.

3. Anomalous Distributed Traffic: Detecting Cyber Security Attacks amongst Microservices using Graph Convolutional Networks

Stephen Jacob, Yuansong Qiao, Yuhang Ye, and Brian Lee. Anomalous distributed traffic: Detecting cyber security attacks amongst microservices using graph convolutional networks. Computers & Security, page 102728, 2022.

## 1.4 Thesis Layout

Chapter 2 provides thorough background information and a literature review for a number of related topics covered in the project, including cyber-security, process mining, microservices, anomaly detection, traffic forecasting and deep learning, particularly the state-of-the-art DL model called the Diffusion Convolutional Recurrent Neural Network (DCRNN). Chapter 3 outlines the first stage of the thesis where process mining techniques and an RNN/LSTM neural network are investigated to learn the general form of sequential process pathways in logged data. In this chapter, a contextual oversight of an application process is provided to predict future events in an ongoing process thread. In chapter 4, an open-source microservices application called DeathStarBench is presented, and applications of distributed tracing to carry out performance management in microservices and a frequency distribution-based anomaly detection method to detect irregular microservice functionality are described. Chapter 5 presents the final stage of the thesis which is a continuation of the work from the previous chapter where the same open source microservices application is executed. For the experiment in this chapter, microservice traffic is modeled by training the DCRNN to learn the spatial and temporal dynamics of the traffic, traffic forecasting is used to predict future microservices activity and threshold based-anomaly detection is applied to detect seeded cyber-security attacks. In chapter 6, the overall conclusions reached over the research project are presented, limitations in regards to the research proposal are outlined and possible future work is discussed.

# Chapter 2

# Background Information & Literature Review

## 2.1 Cyber-Security Attacks

In this section, descriptions of various cyber-security attacks explored for this research project are provided as for their respective countermeasures, and their related literary works.

### 2.1.1 Brute Force Attack

A password guessing, or brute force attack is a form of remote-to-local cyber-attack [1]. This type of cyber-attack is when a hacker attempts to login to an application by continuously trying new usernames and passwords until the correct combination is found through trial and error [2]. This form of attack can take place over a matter of days, months or years. A hacker will typically execute programs to run hundreds of login attempts which could crack an eight character password in a matter of hours [3]. Therefore, a password guessing attack would result in multiple incorrect authentication attempts within a short span of time.

There are two known countermeasures that can be used to prevent a brute force attack: an account lockout policy and the more preferable application of Completely Automated Public Turing test to tell Computers and Humans Apart (Captcha) [4]. An account

lockout policy is where an account is locked out preventing any additional authentication requests after a certain number of failed attempts. This account lockout setting could last over a certain time window. A CAPTCHA application is a challenge and response-based system which authenticates an account using visual test is that is easy for a human being to understand but difficult for a computer program to process.

### 2.1.2 Distributed Denial of Service

A distributed denial of service (DDoS) is a form of cyber-attack where an attacker carries out multiple executions of a computer service to render the service unavailable to legitimate well-meaning users [5]. The magnitude of these attacks is measured in bits or requests per second. DDoS attacks can be categorized as one of the following: a volumetric attack, protocol based or application-layer based.

A volumetric attack is a class of DDoS attacks in the form packet flooding with the aim to saturate the bandwidth of the intended target resource. These packets can be of various protocols including User Datagram Protocol (UDP) and Internet Control Message Protocol (ICMP). A specific type of volumetric DDoS attack is a TCP SYN Flood attack where the perpetrator sends multiple TCP requests to a target server with the malicious intent to overwhelm the target's server so that it is unable to respond to the corresponding acknowledgements. Because the server's gateway is flooded, legitimate users are prevented from receiving connection responses to the server [6]. A countermeasure against the TCP SYN Flooding is the application of SYN cookies, a form of cryptographic hashing in which the client generates a sequence number that is received by the client and observed by the client for verification. Subsequently, the server allocates memory for the verified legitimate client's request [7].

Protocol based DDoS attacks are a class of DDoS attacks designed to consume intermediary resources such as firewalls, load balancers and other server resources. A specific type of protocol based DDoS attack is the remote-to-local SNMP reflection attack where requests with the Simple Network Management Protocol (SNMP) and a fraudulent IP address are distributed to multiple interconnected devices and as those devices reply, the targeted network is brought down from the flood of responses. An SNMP attack can be mitigated using ingress/egress packet filtering provided the network's server is equipped to handle the incoming packet flow [8].

The third variety of DDoS attacks are application-layer attacks which occur over the application layer of a computer system. The intended target of these attacks is the actual application server, where specific vulnerabilities are exploited with the goal of causing the server to crash and fail to communicate with and perform services for its intended users. For this attack, the multitude of requests per second are sent over HTTP connections. These attacks have become one of the most favourable methods of launching a cyber-attack, due to cyber-criminals constantly monitoring and modifying their toolkits to develop new application-layer DDoS threat models. Common forms of application-layer DDoS attacks include Slowloris and a HTTP(/S) Flooding. A HTTP Flooding attack targets both the web server and application-level features. The target server is overwhelmed by the HTTP Flood which is composed of GET or POST requests. Various countermeasures against this attack include the use of a web application firewall [9].

[10] uses Principle Component Analysis (PCA) based anomaly detection (AD) to detect DDoS attacks taking place at the application layer of a system application network. The motivation for this work was due to the immense threat imposed by DDoS attacks towards Internet applications and web servers. For carrying out this approach, instances of user behaviour requesting system resources were extracted from HTTP web server logs to use as data sets. The authors focus on a HTTP GET attacks with flooding behaviours for performing DDoS attacks. Using penetration testing, nine different HTTP Flood attacks were generated amongst user resource requests from a web server for a student resource portal. Different factors included in the testing include the randomness or popularity of requested resources, and embedded objects when attacks were generated.

### 2.1.3 Batch registration of Bot Accounts

A batch registration is defined as the mass creation of bot accounts used for a variety of illegal computation purposes [11]. These bot accounts are typically used by the hacker for dishonest actions like increasing the number of 'likes' on a video on YouTube or a post on Twitter. Other more malicious actions used with the bot accounts include targeting a web application causing a DoS effect, spreading malware or performing fraudulent online activity to sway political opinion.

A possible countermeasure against bot accounts on social media applications noted by [12] is to establish a base threshold of regular online activity and subsequently observe abnormal web requests that indicate the presence of bot account. A possible example of such an indicator is the mass creation of user accounts originating from the same IP address [13].

### 2.1.4 NoSQL Injection Attack

A NoSQL Injection attack is a form of attack where the hacker injects code into a software system to trick the application into behaving in a manner that it was never intended to [14]. A typical NoSQL Injection attack is when a user sends a request to a web application to retrieve the credentials of a single user from a NoSQL database, but the hacker injects malicious embedded code into the search engine box to trick the application to return data for all registered users, read or modify data in the application [15]. A NoSQL Injection can be prevented by avoiding user inputs that have not been sanitized when sending off queries to the database. If JavaScript is needed for said queries, the best practises are to validate and encode all user inputs and a user ensures they are familiar with their language to avoid using vulnerable constructs [16].

### 2.1.5 Man-In-The-Middle Attack

A Man-In-The-Middle (MITM) Attack is a form of cyber-attack in which an attacker intercepts data sent by a legitimate user through eavesdropping or impersonating an authenticated user before the data reaches its intended destination. This is most commonly performed by creating a faux Wi-Fi hotspot. At this hotspot, identity verification is not required and the hacker can get eavesdrop on any information exchanged between the client and the targeted application. A countermeasure to prevent A MITM attack is to use a Virtual Private Network (VPN) where a user's online activity will be encrypted and an attacker will not be able to read the user's private information such as financial information and passwords [17].

### 2.1.6 Cross Site Scripting Attack

Cross-Site Scripting (XSS) is a variety of malicious software injecting attack where malicious coding is embedded in the makeup of a trusted website such as the side script of a web link or a contact form. A typical example of XSS is when unwitting end users send their search query to a server and see the result of their query. However, embedded code could access the victim's privileged information which could result in the user's credentials being stolen like banking information, business emails or healthcare information. Two possible countermeasures against XSS attacks are to filter explicit user input when received based on what input is expected, and encoding user-controllable output data to prevent it being perceived as active content.

## 2.2 Process Mining

Process Mining (PM) is collectively defined as the discovery, analysis and modelling of process information extracted from a data set. This data set is composed of logged task instances which each represent an actual event. Each event belongs to a single process instance or case and is a defined time step of a case. Cases are composed of multiple related events and an event sports a defined set of attributes such as the name of the event and the timestamp of the occurring event. The objective of process mining is to gain a general oversight of operational processes logged in the data set.

Traditional process mining occurs when a set of event logs is fed into a process mining application tool, e.g. ProM [18] as input, and an output process model is returned. This generated model is visible to the user via the process mining tool, and the modelled processes highlight the sequential execution paths recorded by the event log data, supporting analysis of said processes by the user. This form of process mining is called process discovery and is shown in Figure 2.1.

The model can also be used to carry out conformance checking to determine if a newly introduced process instance conforms to the generated model. If the sequential ordering of a particular case of events does not comply with the returned model, the said case would be considered a deviation from the model's architecture. A third purpose of

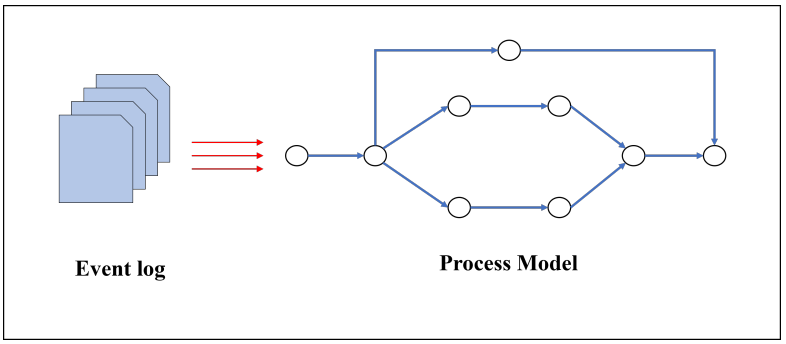



FIGURE 2.1: Example of Process Discovery

process mining is to improve the said process mining result called process enhancement [19].

The application of process mining to visualize and monitor the process pathways of an application process, and discover cyber-security issues in the resulting process model has been used before as shown by the works of [20], [21] and [22].

The work by [20] studied the application of an Intrusion Detection System (IDS) which produces an immensely high level of security alerts. The motivation for this is that cyber security personnel would have a difficult time managing the numerous alerts and distinguishing the more annoying innocuous alerts from the more serious ones. This paper proposes a four-step process mining approach to discover attack strategies in a data set of logged alerts returned by an IDS. After process discovery was performed by the process mining tool and a process model was returned, an expert analysis of the generated model was carried out and major observations were made. [22] also employed process mining to extract information from IDS alerts with the aim of discovering an attacker's behaviour and the multistage attack strategies they adopt. These attack strategies would be presented to network administrators using user-friendly visual models. The authors for this work also discovered attack strategies that were rather large and complex to visualize and comprehend. Therefore, they also employed hierarchical clustering to cluster larger models into smaller and simpler attack models. Both of these approaches were tested using a real-life data set of IDS alerts from University of Maryland.

[19] employed process mining to analyse audit trails, detailed records of events occurring within a system or across a network. The authors believed that businesses will need to increasingly store and monitor audit trails for their system application. Audit trails could be used to provide a logged trace of all user/system actions or events with labeled

terms for the purpose of discovering the cause of detected security events. Audit trails could also inspect certain deviations of system activity. This paper explores process mining techniques, specifically the $\alpha$ algorithm, and focuses on two existing problems: the detection of anomalous process instances not classified as violations that can be used to update the existing model and conformance checking where new instances are tested against the model and determined if they conform to the model.

The work by [23] proposes process mining to generate a process models represented as a Petri-Net with an event log of workflow processes as input. In terms of process mining, a case in this work would correspond to a workflow instance and an event refers to a well-defined step in the workflow process instance. To generate a Petri-Net, the authors employ the $\alpha$ algorithm, and highlight its advantages and limitations. The authors for this paper have applied their process workflow mining techniques to two application event logs to mine the different processes involved. The first process represents a visitor's log of multi-disciplinary patients to different medical specialists, and the second model illustrates the logging of fines collected by the Dutch Judicial Collection Agency.

[21] describes how process mining can be used to monitor process execution paths of computer programs by observing the sequential processes of usage data. The inspiration of this approach is that attacks such as a buffer overflow or an attack where a user is tricked into using a program that was not intended for by the developer would not be easy to detect. One feature such attacks have in common is that they alter the sequential process paths in an application. This paper aims to visualize these altered execution paths using constructs called Petri Nets. Using these structures, irregular process paths could be identified by comparing them to a process model composed of regular process workflows.

## 2.3 Deep Learning & Neural Networks

Deep learning (DL) is defined as the application of a set of machine learning tools called **Neural Networks** (NN) or **Artificial Neural Networks** (ANN). ANNs are used to process data in order to capture and learn special meanings of the input data series through representational learning and output resulting data.

The ANN is modeled after the human brain which processes input signals over a number of layers containing a set of distribution nodes. The actual NN model is comprised of one or more layers each with their own function. The first layer of the model inputs the data, the middle hidden layers process the data and the final layer of the stack provides the generated output values in response to the input. The term 'deep neural networks' (DNN) refers to the number of layers that compose the model. A basic architecture of an ANN is displayed in Figure 2.2. This DNN architecture can applied for a variety of topics including speech recognition, natural language processing, image recognition and machine translation.

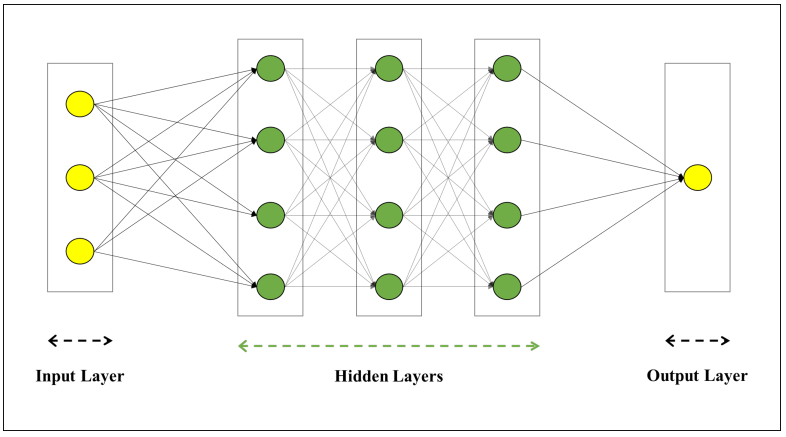



FIGURE 2.2: Architecture of a Deep Neural Network

### 2.3.1 RNN & LSTM

Recurrent Neural Networks (RNN) are a class of NNs where the output of the model is fed back into the model network over a number of iterations. In an RNN, the connections between the nodes within the layers form a graph structure along a temporal sequence. This structure supports the learning of sequential data and temporal dynamic behaviour which makes the RNN applicable for tasks like process modelling and handwriting recognition [24]. The learning of temporal dependencies by a RNN is possible through an algorithm called backpropagation through time (BPTT). The RNN inputs one temporal time series at each time step to calculate inherent errors and update weights [25].

A special class of RNN is a Long Short Term Memory (LSTM) [26]. In sequential tasks, for example, speech or handwritten recognition, long-term dependencies are typically inherent in the data series. Such dependencies at certain time steps in a series can impact the remainder of the data. An LSTM is trained to iteratively capture and learn

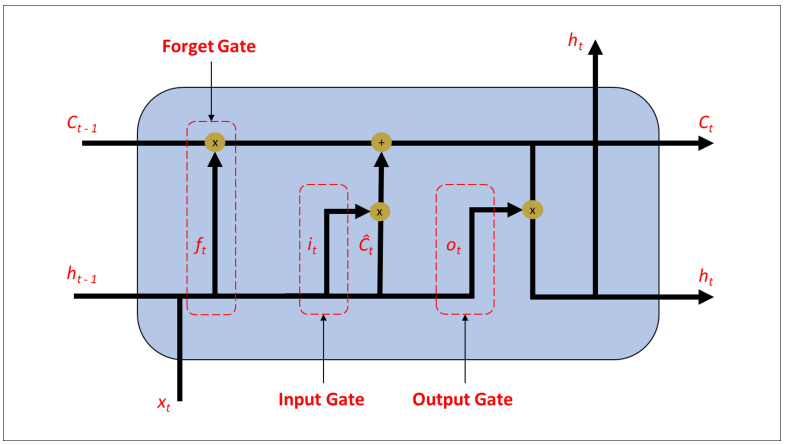



FIGURE 2.3: Architecture of an LSTM Node

such dependencies in the input data set over a time series. In an LSTM model layer, the computation nodes are now comprised of activation functions and a sub-network of units which regulate the flow of information that flows through the model. These units which capture these arbitrary dependencies are referred to as a input gate, a forget gate and the output gate. The architectural structure of this LSTM unit and other components are displayed in Figure 2.3.

When an LSTM is training, data is entered into the layered network over a temporal series. Given an actual input value $\mathbf{x}$ at time step $t$, the LSTM cell state denoted as $C$ represents a compiled aggregation of all processed values entered into the cell over previous time steps. These actual gates control how data from the previous time step $t-1$ determine the current LSTM state $C_t$. The LSTM cell unit itself is entered into and processed by the actual LSTM unit via the input gate $i_t$. The forget gate denoted as $f_t$ determines which data at current time step is to be discarded based on the previous output state $h_{t-1}$. Finally, the output gate determines the actual output value of the LSTM cell, denoted as $o_t$. Each gate operates its own activation function to map the input values to their respective output values.

The formulae for each of these gates, the LSTM cell state $C_t$ and the output cell value $h_t$ are defined in Equation 2.1:

$$
\begin{aligned}
f_t &= \sigma(W_f * [h_{t-1}, \mathbf{x}_t] + b_f) \\
i_t &= \sigma(W_i * [h_{t-1}, \mathbf{x}_t] + b_i) \\
o_t &= \sigma(W_o * [h_{t-1}, \mathbf{x}_t] + b_o) \\
\hat{C}_t &= \tanh(W_C * [h_{t-1}, \mathbf{x}_t] + b_C) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
h_t &= o_t * \tanh(C_t)
\end{aligned}
\tag{2.1}
$$

where $W_f$, $W_i$, $W_o$, $W_C$ and $b_f$, $b_i$, $b_o$, $b_C$ are the weight matrices and biases respectively for the forget, input, output gates and the LSTM cell state $C$ respectively. Given this LSTM cell processing, LSTMs are suited for modeling data series composed of sequential based processes instances.

In the field of business process management, predicting process behaviour is an important necessity. [27] proposes a LSTM be used to monitor and learn the typical form of business process cases within a data set and subsequently predict the remaining continuation of ongoing cases, as well as the remaining time of said case. The authors investigate different LSTM architectures applicable for exploring the specified tasks. The authors also test their approach against four available data sets and show that it outperforms a previous methodology by [28]. The authors also specify the requirements of building and developing such models like trial-and-error experimentation and appropriate tuning of the models as the accuracy of model's prediction correlates with the size and parameters of the data set.

The research article by [29] proposes LSTMs to predict a subsequent event in a running process instance. This proposal was motivated by the application to Natural language Processing (NLP). The aim for this paper is to show that when a DL model learns the explicit form of process instances, the model can be substitute a defined process model. The methodology proposed by this paper was shown to outperform the best approach towards subsequent event prediction by [30].

### 2.3.2 Graph Neural Networks

#### 2.3.2.1 Convolutional Neural Networks

Another class of DNN/ANN is the convolutional neural network (CNN). This type of ANN is applicable for learning visual image and graphical-based data series. CNNs apply semantic segmentation to identify the features in each pixel of an input image value. Therefore, this class of ANNs are well suited for classification and recognition of data values featured on both image and video content, and computer vision [31].

The hidden layers of a CNN are typically composed of a set of convolutional layers that take the visual data as input and perform a convolution operation where various objects are assigned learnable weights and biases to generate a feature map which is passed on to the subsequent layer. Another component of the CNN layer is a Pooling layer that reduces the spatial dimensionality of the convolved features to reduce the computational power required for the network model to run. CNNs can be trained to learn both regular Euclidean graph and/or pixel data without needing to extract manual features. However, CNNs are limited in that while they can model graphical constructs with an underlying grid structure, they are not suited for modelling non-Euclidean graph structures. The architectural design of the CNN is displayed in Figure 2.4 where an image vector displaying a numeric value is processed by the network and returns the number four as output.

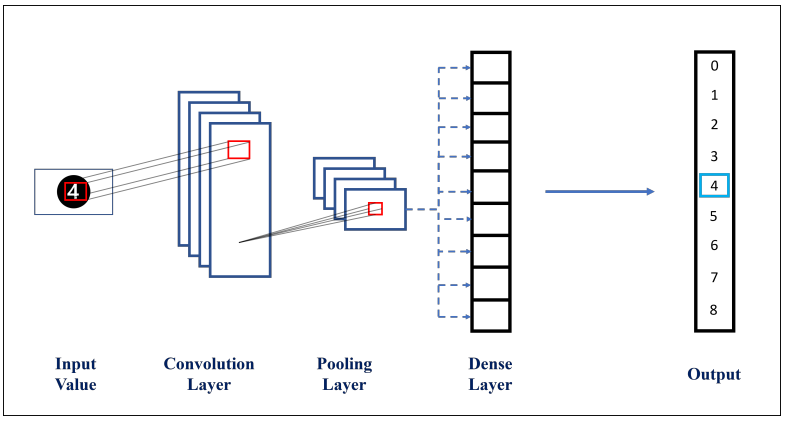



FIGURE 2.4: Convolutional Neural Network

The work by [32] notes the effectiveness of CNN models for solving a variety of machine learning problems, but highlight that they are not suited for modeling structures with diverse dimensions and non-Euclidean domains, e.g. a chemical molecular data set.

The authors proposed a more general and flexible GCN which inputs arbitrarily shaped graph-structured batch data with their evolving residual graph Laplacian which is to be trained in a supervised fashion. The model would be able to learn the graph structure for each data sample that optimizes the hidden-node connectivity. Experiments for this paper show that this approach with the evolving GCNN outperforms state-of-the-art methodologies on several data sets.

[33] presents the Diffusion Convolutional Neural Network (DCNN), a new CNN model designed for learning how to represent a diffusion-convolution operation processes using graph-based data. The main motivation behind this work was to provide a representation that encapsulates a diffusion process across nodes as a basis for a prediction problem. The DCNN model was trained using a GPU server and its performance for node classification tasks was explored, and results show several advantages in terms of accuracy, speed and flexibility. The model was evaluated against five probabilistic relational models and kernel methods and was shown to outperform all these different methods.

#### 2.3.2.2 Graph Convolutional Neural Networks

Graphs are comprised of an aggregation of nodes interconnected by set of edges. A graph can have either a Euclidean based structure, or a non-Euclidean structure where the number of node connections, levels of adjacency and neighbourhoods are more unordered, for example, a social networking site of interconnected users. Because CNNs require an ordered grid-based structure to model graph-based data, it is not feasible for them to model the more arbitrary design and space of the social network. To model and exploit the more arbitrary architecture of non-Euclidean-based graphs, a more generalized version of CNN called a Graph Neural Network is required [32].

Graph structures utilize local relations between many individual nodes, which GNNs exploit to perform given tasks such as node classification and prediction [34]. GNNs are also used to model several real-life structures such as the brain and nervous system, and inter-molecular relationships among chemicals [35]. Furthermore, they are used to perform several everyday tasks including learning representations of molecular fingerprints [36], traffic network forecasting [37] and NLP tasks [38].

Figure 2.5 displays the general architecture of a GNN. Given an input graph state representation, the input value is fed into the GNN to model and learn the graph-based features of the data. In this GNN, a non-linear *ReLu* activation function to separately activate the nodes of the networks hidden layers, and process and map the input values to output graph state representations. Finally, the output graph state is returned after the GNN is trained.

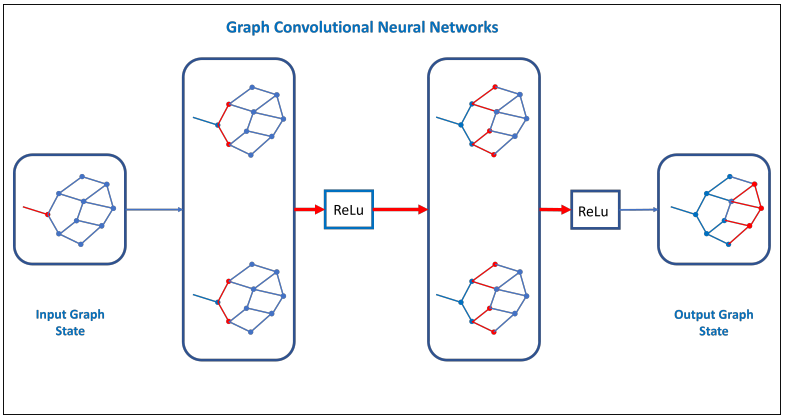



FIGURE 2.5: Graph Convolutional Neural Network

A survey written by [39] provides a comprehensive review of various graph convolutional neural networks. The study introduced two classifications of existing GCNN models: spectral-based GCNs and spatial-based GCNs depending on the type of convolutions. A GCNN with a spectral construction draws on the properties of the convolutional layers that comprise the model, such as the spectrum of the graph Laplacian and the adjacency levels of the domain. The first spectral-based GCN was proposed by [40] which shows that it is possible to learn from convolutional layers independent of the input size without hindering the testing error. Spatial-based GCNs are models proposed by [41] that have the ability to learn the spatial or geometric features of graph-based data including the ordering of node neighboring and positions.

#### 2.3.2.3 DCRNN

Spatial and temporal traffic forecasting is a nascent research topic with applications in process diffusion or road transportation domains. Networks that represent said domains can be illustrated using graph-structured models. The works by [42] and later [33] propose new CNN-based models to incorporate these approaches.

[42] presents the Diffusion Convolutional Recurrent Neural Network (DCRNN), a state-of-the-art model that incorporates the learning of both the spatial and temporal dependencies along a time series of traffic-based data. The motivation for this paper was to predict future traffic activity along road networks. The authors for this paper study the traffic forecasting problem and model the spatial dependencies of traffic as a bidirectional diffusion process along a directed graph within a RNN. The DCRNN model was tested using two different data sets of recorded real-world traffic and the proposed approach was shown to outperform multiple state-of-the-art baseline methods. Another contribution made by the authors was that the DCRNN model is not limited to road network-based transportation traffic.

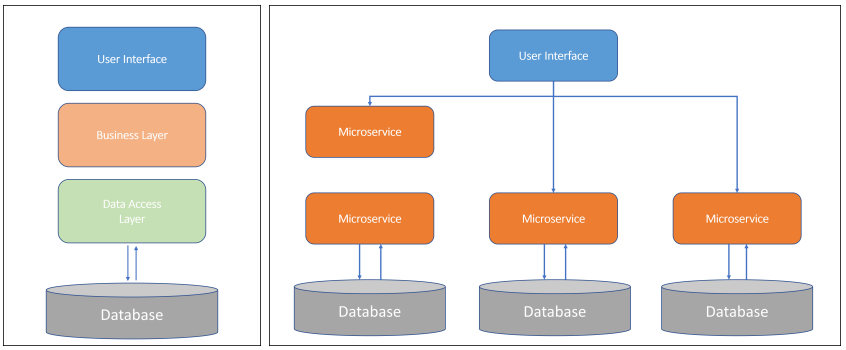
[43] employs the DCRNN model to perform traffic forecasting to predict future traffic loads on the links of a telecom network. The motivation for this work is that due to the increasing complexity of telecom networks, it has become correspondingly important to improve the efficiency of the network infrastructure such as network management and resource allocation. These requirements can be met through traffic prediction. The authors' aim is to predict subsequent loads on a link to telecom network given previously learned representations of traffic loads. Firstly, a DCRNN model was trained to capture the topological features of the telecom network links using a real telecom data from a backbone network. This methodology was compared against and shown to outperform four existing baseline approaches.

[11] stresses that due to the numerous amount of real-time unstructured data in microservices, it is challenging to enforce security mechanisms for a microservice application. This paper proposes the use of irregular microservice traffic detection for RPCs to discover anomalies in the data produced by a dynamic microservice production scenario. The proposal by this is a two-stage process called Informer. In the first step, a density-based clustering technique DBSCAN [44] to discover correlated RPC chains which are subsystems of the applications, and then train a GCNN model, the DCRNN [42] to learn the spatial and temporal dependencies of each RPC chain and use traffic forecasting to detect irregular RPC traffic. To demonstrate the effectiveness of the Informer approach, the Informer technique was evaluated using two real-world data sets which contain malicious RPC traffic from two different forms of illegal computing activity: a batch registration of bot accounts and a cyber-attack to crack an account.

## 2.4 Microservices

Microservices, or the microservices architecture (MSA), are a service-oriented software architectural design based on a distributed system where the whole application is decomposed into several smaller components called *micro*services. These *micro*-services are each responsible for a single unique process of the application's overall functionality e.g., a user logging into their registered account, sending an email or message to a fellow user, or storage of data objects.

In a microservices application, a single microservice is a well-defined interface that can be called in response a user's RESTful or HTTP API call to perform specific business functionality. In said distributed application, the individual microservices can communicate with other over a shared cross-service API, which supports requests for functionality that span across several microservices. The individual microservices also run alongside each other in the application but can be developed, scaled, tested and deployed independently due to their polylithic design. This makes microservices particularly advantageous over the monolithic application design which is composed of only a single tiered architecture which must be developed and scaled in its entirety. Due to this, microservices are gaining traction as the latest software design platform that is being adopted by many commercial software enterprises including Twitter, Netflix, Amazon and eBay [45]. The single-tier design of a monolithic software architecture and the polylithic design of microservices are displayed in Figure 2.6(a) and Figure 2.6(b) respectively.




(a) Monolithic Architecture

(b) Microservices Architecture

FIGURE 2.6: Monolithic vs Microservices

The motivation for the work by [46] was the major shift from the monolithic architecture design to microservices for several end-to-end service applications in recent years. This

paper explores the microservice architecture (MSA), their appeal and the implications the MSA has on cloud application server design. For this work, the authors present a new end-to-end microservice application comprised of tens of microservices that implements a movie streaming, renting and reviewing system is presented. Various microservices for the application include the functionality to display movie information, add a movie review or renting a movie review. To study the effects microservices have on the design on data center hardware, the authors measured the time spent in application computation versus the communication between the microservices RPCs and quantified the I-cache pressure produced by the MSA.

The paper by [45] explores the effect of microservices on the cloud system stack from hardware acceleration to the actual application design, as well as the Operating System (OS) and networking overheads, cluster management and framework design. To study the miMSA, the paper presents an open-source testbed suite composed of six microservices based applications called DeathStarBench. Various end-to-end services that compose the testbed include a social networking service, a movie review service, banking system and an e-commerce banking system. The authors list the design principles for the benchmark suite including the heterogeneity, modularity and end-to-end operations. Finally, the paper highlights that the methodological disadvantage of client-server applications cannot rely on the client to report performance issues like QoS violations which is rectified by implementing distributed traces to determine which microservice is the cause of the issue.

[47] proposes a SaaS called *FlowTap*, a monitoring and policy enforcement infrastructure for network traffic to secure a cloud application. This work is motivated by two major security challenges for microservices: the complex application design activity making the actual monitoring and securing networks rather challenging and that if one microservice component was compromised, the application as a whole could be potentially compromised. In this paper, the authors propose and evaluate their SaaS which aims to provide system administrators to construct a global view of their applications even when the application is distributed over the cloud. As part of this work, an empirical study showed that the API primitive *FlowTap* supports various monitoring scenarios and policies. The end result of this paper is allowing applications to leverage the solution to deploy security monitors to detect and block threats to their network.

The work by [48] provides a deep level description of the MSA, and outlines the security challenges concerning a microservices based architecture. The paper looks at different structure levels of a microservice-based architecture concerning a software development life-cycle whether it be composed of components, architecture, infrastructure or the governance. The article determined that microservices require secure measures at said levels. At the infrastructure level, a firewall can be implemented to filter outgoing and incoming traffic, or an IDS. On the component level, a user API request call to a microservice should only be sent after authentication and authorization has taken place i.e., verifying an access token. If the token was verified at the gateway level, the microservice request would become vulnerable to the confused deputy problem [49].

[50] provides a survey describing security risks that pose a threat to fog applications with a microservices based architecture. The focus of this survey include security issues and solutions that rise in microservice communications in regards to four aspects: cloud containers, data, permission and network. The security issues of Docker containers include potential poisoning of the Docker images, DoS attacks and escapes from containers. [51] implemented *cgroups*, a Linux feature that safely create virtual environments that monitors the application for containers using multilateral balanced security in multi-tenant applications to prevent DoS intrusions and keep the application secure. A data-related security issue in microservices is when data could be intercepted over the microservices' immensely complex design. Attackers could also infer information about the business operation functionality from the intercepted communications. [52] proposes a data encryption method to store data on cloud servers using a combined hierarchical identity-based encryption (HIBE) and ciphertext-policy attribute-based encryption (CP-ABE) to keep data confidential from untrusted cloud server providers (CSP). The use of ineffective access control or permission mechanisms are a potential security concern for applications in a distributed computing environment. To mitigate manipulations of network resources, [53] proposed a trust framework to authenticate network applications and set authorization restrictions on network resources. [54] presents a microservice application using the OAuth standard for access delegation to guarantee the security for access control. The fourth security issue, network security, is particularly concerning in regards to microservices due to the frequency of communication between the individual service components. Various attacks that can be simulated include DDoS, Man In The Middle (MITM) and Address Resolution Protocol (ARP) Spoofing.

## 2.5 Distributed Tracing

Distributed tracing is defined as both the monitoring and logging of the execution path of related API calls within a distributed application. In the application, a user's API request to perform the required functionality will typically span across multiple services. An application is configured to use a distributed tracing tool and an application's request call will return a **distributed trace**, a record of the logged execution path throughout the application.

A distributed trace is represented as a sequence of spans. One span represents a recorded API call to a single distributed application event. In a trace, each span will share a unique identifier representing the existing distributed trace and user's API call the span belongs to. Other attributes sported by spans include the name of the service call executed, the timestamp and duration of the service call, as well as meta information such as the HTTP URL executed by the triggered microservice operation and the response code to the HTTP call. A distributed trace will record all services called by the API call sent to the application. Throughout a trace, spans will have various parent-child relationships, where parent spans can have multiple child spans and a child span can have only one parent. Given a distributed trace composed of generated spans **A**, **B**, **C** ... **G**, a trace tree anatomy can be derived which is shown in Figure 2.7.

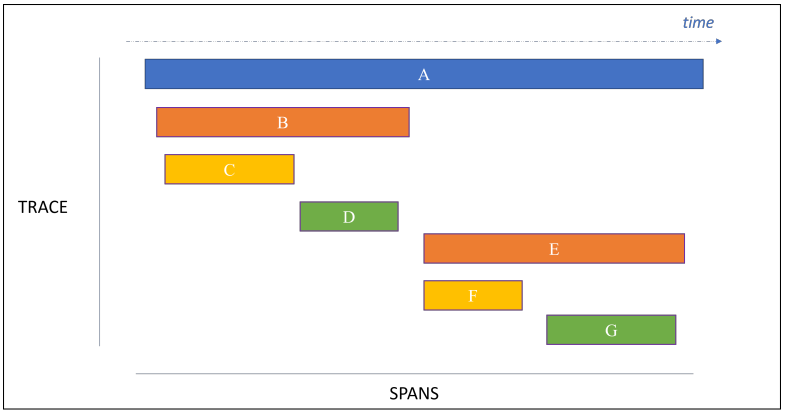



FIGURE 2.7: Tree Anatomy for a Distributed Trace of Spans

The trace tree representation for the distributed trace illustrated in Figure 2.7 displays the distributed topology amongst the application services represented by each span. In the tree anatomy, the span labelled **A** is the first span generated which takes place over the entire life span of the distributed trace. **A** makes an API call to a child service generating span **B**, so spans **A** and **B** have a parent-child relationship. Likewise, for

the duration of span **B**, spans **C** and **D** are generated concurrently and thus are the child nodes for **B**. Finally, the remainder of the distributed trace is composed of span **E**, another child node of span **A**.

Distributed tracing is typically used to provide a vital comprehension of the behaviour of distributed system applications, as well as the user's API requests that propagate throughout the system application. Information returned by a distributed tracing tool include causal dependencies between the distributed components The application of distributed tracing is commonly used for debugging and performance management of a microservices-based application.

The motivation for [55] is that distributed tracing can be a difficult process due to its complexity and the application specifications, and a lack of support tools for abstracting, navigating and analysing the tracing data. This paper proposes the use of tracing data to extract metrics, dependencies with the aim of detecting anomalies and patterns in the data. All prototype tools for this approach were published while conforming to the OpenTracing standard and developed anomaly detection standard. This approach also identified limitations for OpenTracing which could be useful for future researchers specializing in building distributed tracing tools and standards.

[56] investigates the application of distributed tracing to improve the observability of faults in serverless applications. Such faults in various applications are unpredictable and can occur in multiple points, even in simple compositions and developers often have to rely on ambiguous error messages and scattered logs for root cause analysis. The contributions for this paper are a serverless fault observability model and a first instantiation of the model based on serverless platforms like AWS Lambda and OpenWhisk, and a prototype implementation and evaluation of two serverless tracing approaches with the aim of improving fault observability.

[57] provided a qualitative study to gain an understanding of the practises, advantages and challenges of monitoring distributed systems. The present challenge for monitoring a cloud service application is due to the dynamic and complex modular design of such applications. The study showed that there is no common solution for all cloud companies despite multiple attempts to bridge this noticeable gap. The authors conducted an industry interview amongst various stakeholders in monitoring software providers,

DevOps engineers and consultants. The study highlighted various questions for stakeholders in distributed monitoring and what the different technical and organizational strategies for various companies are.

## 2.6 Anomaly Detection

Anomaly detection, or outlier analysis, is defined as the detection of irregular instances or outliers that occur within a data series. These unusual events discovered in said data series are classified as anomalies by the application of anomaly detection if their behaviour or form does not conform to that of the majority of the data. A 2-D graph illustrating normal activity and irregular anomalies amongst a data series is displayed in Figure 2.8. Real-world examples of outliers include the detection of fraudulent insurance, the presence of cyber-attacks, or irregular enemy activity by military surveillance.

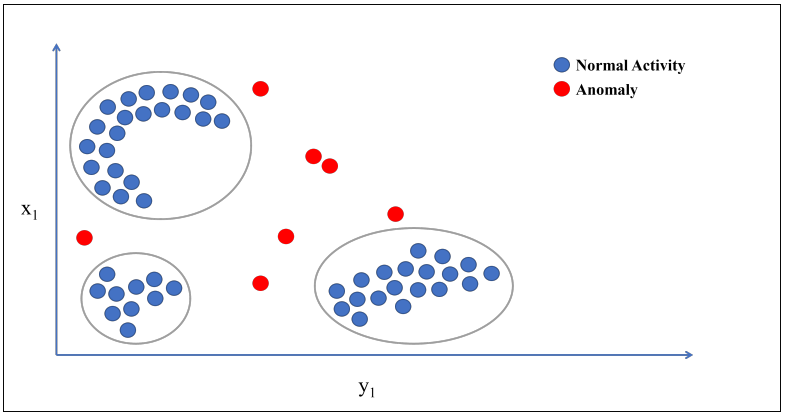



FIGURE 2.8: Detected Anomalies in a Data Series

Anomaly detection can be categorized into two different classifications. The first class is defined as pointwise anomaly detection and is the most recognizable form of anomaly detection where the detected outliers are represented as single independent instances within a data set. The other class is defined as group anomaly detection (GAD) [58] which is a more recently emerging form of anomaly detection. By applying this form of anomaly detection, the detected anomalies within the data series are represented as collections of instances instead of individual outliers.

Anomaly detection is performed using two different means: supervised and unsupervised anomaly detection [59]. When supervised anomaly detection is applied, both regular and irregular data instances are labeled. A machine learning model is then typically trained

to learn the general behaviour of the data and then leveraged to detect outliers in the data series. This outlier detection method is also used for classification tasks [60]. By contrast, unsupervised anomaly detection is when the actual data series lacks specified structure and data labels. It is implied that normal data is more frequent in the testing data series and therefore the machine learning model is more robust to anomalies in the testing data. An issue of the supervised approach is that irregular outliers are far fewer in the training data set than in the testing set. A proposed solution to this problem is injecting anomalous instances into the training set to increase robustness of the machine learning model when detecting outliers [61].

[62] presents how the ProM framework [18] can perform anomaly detection in Process Aware Information Systems (PAIS) logs to facilitate the automation of their business processes. In companies, however, a rapid response to changes in process strategy is often demanded due to flexible PAIS compromising the security of the systems as users' actions can cause violations in financial loss. In this work, process discovery of PAIS was carried out, followed by the filtering of specific models, and the most appropriate model in terms of structural simplicity and process behaviour was selected. This methodology is based on the control-flow perspective and determines a process instance as anomalous if it does not conform to the selected model. The authors tested their approach using a real-world log of an information system for a Dutch municipality.

The aim of [63] was to detect anomalies based on distributed traces which contain detailed information on the services provided in order to detect faults and issues of services. The detection of anomalous processes within large scale distributed systems was deployed on heterogeneous hardware and has multiple scenarios of normal operation where it becomes difficult to detect such anomalies. The authors address these issues by applying response time anomaly detection with a focus on unsupervised learning and deep learning techniques, including an applied dynamic error threshold, a module for false positive rate reduction and a descriptive classification of anomalies. This model would be composed of a combined Gated Recurrent Unit (GRU) and variational autoencoder (AEVB) [64]. Experiments were performed with real-world production tracing data with artificially injected anomalies.

[58] uses deep generative models (DGM) to perform an anomaly detection problem by detecting irregular collections of grouped patterns that do not conform to normal

patterns in a data series. The authors propose a generative methodology using the following models: an adversarial autoencoder (AAE) and a variational autoencoder (VAE). Both models are used to detect pointwise anomalies where group memberships are a known priority. This approach was tested using a variety of synthetic and real-world data sets to demonstrate its effectiveness and was shown to outperform a number of state-of-the-art GAD techniques. The results of the authors' experiments demonstrate that the approach is effective and robust in detecting grouped anomalies. The authors determined that DGMs can outperform state-of-the-art GAD techniques though they require a large number of grouped observations for model training.

[59] provides a survey of anomaly detection techniques used in various literature, present a general description of anomalies. Based on underlying approaches, the authors divided said AD methodologies into the following categories: classification, nearest-neighbour, clustering, information theoretic, spectral and statistical techniques. The advantages and disadvantages for these categories were outlined. Furthermore, the authors briefly describe the different application domains of which anomaly detection techniques have been applied e.g., cyber intrusion detection, image processing or fraud detection. The motivation for this survey was to provide a clear understanding of the different directions where research has taken the topic of anomaly detection. Classification-based anomaly detection occurs when a model is trained to learn a set of labeled instances and is evaluated against a testing data set to classify the testing instances as normal or anomalous [65]. Nearest-neighbour-based anomaly detection is based on the assumption that normal instances occur within densely populated spaces while anomalous instances operate far away from their closest neighbour. A data instance is defined as anomalous as defined by its distance from its $k^th$. The distance metric can be a simple matching coefficient, but more complex distance metrics can be used [66]. In clustering-based anomaly detection, data instances are grouped into clusters primarily using unsupervised or semi-supervised data with the assumption that anomalous instances do not belong to any cluster centroid. A clustering technique called Self-Organized Maps (SOM) [67] has been applied to perform anomaly detection in applications including intrusion detection [68] and fraud detection [69] using a semi-supervised data model.

## 2.7 Conclusion

In preparation for this research project, an extensive literature review over a variety of fields was carried out and relevant background information was also provided. The different fields covered include various forms of cyber-security attack, the microservices architecture (MSA), the application of distributed tracing, anomaly detection, process mining, and deep learning.

One of the objectives for this research project was to enhance cyber-attack prevention. One proposed means to do so was to gain a contextual oversight of an application process to discover the goals, services, and infrastructure of the process. This discovered oversight would prompt cyber-security personnel to identify and prioritise critical assets for their enterprises' software application and thus pre-empt impending cyber-threats to those assets. Two possible options to discover this contextual oversight include the applications of process mining and deep learning. Various works concerning the field of process mining highlight that process mining tools can take as input a data set of logged events and output extracted process models. These output models make explicit the execution paths of the process flow in the event log, and thus discovered and analysed the underlying behaviour of existing processes. The literature review over deep learning proves that this contextual oversight of an enterprise application can be gleaned by training and compiling a deep learning model, specifically an LSTM model, to internally learn the sequential form of input data, and subsequently predict future events in ongoing process threads.

Due to the appeal of the microservices architecture (MSA) and recent attacks on software enterprises which have adopted the MSA, an extensive literature review of microservices was included. A number of these works feature an open-source benchmark suite of microservices applications called DeathStarBench, which was a suitable testbed for executing microservices and gaining a vital comprehension of the MSA. This review also highlights potential security risks posed towards microservices, particularly the conundrum that if one microservice was compromised, the cyber-threat could propagate throughout the application due to their collaborative functionality. Distributed tracing was included as it was necessary to configure applications to monitor and log the execution path of microservice calls propagating throughout the application for performance management.

The second research objective for this project was that anomaly detection approaches could be used to detect seeded cyber-attacks in a microservices application, so literary works based on anomaly detection were included. One anomaly detection problem from this review was based on detecting grouped patterns in a data set. In this project, cyber-attacks were simulated against the application generating irregular quantities of microservices calls logged using distributed tracing. An anomaly detection method could be applied to detect the grouped distributed traces. To learn what cyber-attacks could be detectable by this anomaly detection problem, an extensive review of cyber-attacks was included. Cyber-security attacks studied for this project include a Brute force password attack, a batch registration of bot accounts and a Distributed Denial-of-Service.

In the literature review for deep learning, a set of neural networks called convolutional neural networks (CNN) were outlined, particularly a state-of-the-art graph CNN called the Diffusion Convolutional Recurrent Neural Network (DCRNN). This DCRNN was the subject of several literary works for modelling an existing traffic network model augmented with active, ongoing traffic flow, e.g. a car traffic on a road network. This DCRNN model was notable for being able to learn both the spatial relations of the model's architecture and the temporal dependencies of the traffic thus capturing the state of the traffic over a specified time series. In the final part of the thesis, a DCRNN model was trained to learn such dependencies of existing microservice traffic on a running application and subsequently make predictions of microservice activity. The grouped anomaly detection problem was applied to these predictions.

# Chapter 3

# Improving Cyber Situational Awareness through Application Process Flow Prediction

## 3.1 Overview

Due to business organizations being constantly targeted by increasingly advanced cyber-security attacks, similarly sophisticated countermeasures against such cyber-attacks are being proposed by software developers daily. One proactive countermeasure is being able to predict future events of ongoing business application threads by discovering a contextual oversight of the enterprises' software application process. This provides cyber-security personnel with improved cyber situational awareness and helps to prioritise critical assets and pre-empt impending cyber attacks targeting these assets.

This chapter introduces two methodologies for carrying out this proposal. The first approach is the application of conventional process mining algorithm to generate a process model which illustrates an organisation's business process and makes explicit the aims, services, and infrastructure of the general process. These elements greatly enhance cyber-security risk assessment. The second approach is to train a deep learning model to memorize the typical form of sequential process threads, thus gaining a contextual oversight of the overall application process and evaluate said model by predicting future events in ongoing threads. The deep learning approach is the primary contribution for

this chapter. Two existing methodologies for training the deep learning model are also presented: 1) a prefix-based method used by [27] and 2) a teacher forcing methodology used by [70].

Building on the proposed approach above, and the two techniques described, this chapter will address the following research question.

*Can process mining and deep learning models predict next steps in an application process?*

## 3.2 Process Mining Approach

One approach to gaining a contextual oversight of a general application process workflow with the goal of improving cyber situational awareness is the application of process mining. Through the use of process mining, a process model is returned which makes explicit certain elements of an application process including the aims, services, execution path and framework, which greatly enhance the risk assessment of cyber-threats.

For this approach, an open-source process mining tool is presented, two selected process mining algorithms are explored and a data set composed of multi-stage cyber intrusion alerts is present which is fed into the process mining tool and used to return a process model representation of the cyber-attack scenario. By reconstructing this cyber-attack, the goal is to gain a contextual oversight of the process workflow for the cyber-attack to gain satisfactory cyber situational awareness.

### 3.2.1 ProM

ProM is an open-source process mining framework that allows users and developers to work with and apply process mining techniques in the form of plug-ins. ProM is extensible and easy to use. ProM provides process mining support not only for process discovery, analysis and conformance checking but for different data attribute perspectives such as time, resources and control flow. The tool also provides key data metrics including the number of event types, cases and processes within the event log. ProM is a world-leading tool in the field of process mining, [71], is implemented in Java, platform independent and available at [72].

### 3.2.2 Alpha Algorithm for Process Mining

Here, a process mining tool called the $\alpha$ algorithm [23] is described which can be used as an available plugin for the ProM tool outlined in Section 3.2.1. The algorithm is used to return a process model that illustrates the workflow perspective of a process. The aim of this section is to give a formal explanation of the $\alpha$ algorithm and describe the application of the algorithm using a sample event log found at [73].

Given an input event log of process instances, the $\alpha$ algorithm outputs a process model called a Petri-Net [74]. Petri-Nets are a simple graph construct that support parallel and distributed processes. The primary component of a Petri net is a place which represents resources, states or conditions of a process. The states of these places can be changed via transitions and are all interconnected via arcs. A specific variety of Petri-Nets are *WorkFlow nets* (WF-nets) which are tailored to a workflow-based process and where all existing nodes are on a path from a source place to a sink or destination place.

Given a case of events $\sigma = e_1, e_2 \dots e_n$ and the sample event log $W$ illustrated in Table 3.1 as follows:

| CaseID | Event |
|---|---|
| 1 | A |
| 2 | A |
| 3 | B |
| 3 | D |
| 1 | C |
| 1 | E |
| 4 | B |
| 2 | E |
| 4 | F |
| 2 | C |
| 2 | G |
| 4 | D |
| 1 | G |
| 3 | F |
| 3 | G |
| 4 | G |

TABLE 3.1: An Example of an Event Log

it can be inferred that the event log has a vocabulary of 7 event types and there are four existing cases.

In the case of the $\alpha$ algorithm, every pair of events is required to have an ordering relation between them. When the $\alpha$ algorithm is applied, four different relations are extracted between every two events in a case. Let $T$ be a set of tasks or events and $W$ be an event log where two events $a, b \in T$ and the four relations $>_W$ (follows), $\rightarrow_W$ (causal), $\|_W$ (parallel) and $\#_W$ (unrelated) are specified as follows:

- $a >_W b$ if and only if there is a case $\sigma = e_1, e_2, e_3, \ldots e_{n-1}$ in $W$ such that $\sigma \in W$, and $t_i = a$ and $t_{i+1} = b$
- $a \rightarrow_W b$ if and only if $a >_W b$ and $b \not<_W a$
- $a \|_W b$ if and only if $a >_W b$ and $b >_W a$
- $a \#_W b$ if and only if $a \not>_W b$ and $b \not>_W a$

Given the four traces defined in $W$, and abiding by the set of established relations, it is inferred that the two sequential orderings $A >_W C$ and $A >_W E$ and the parallel relation $C \|_W E$ hold true in the event log.

Given the event log $W$, the application of the $\alpha$ algorithm is denoted as $\alpha(W)$ and is formally explained in the following steps:

1. analyze the event log $W$ to discover all transitions $T_W = t \in T \mid \exists_{\sigma \in W} t \in \sigma$ that occur and correspond to an event
2. create a set of output transitions $T_I = t \in T \mid \exists_{\sigma \in W} t = \mathit{first}(\sigma)$ for every event at the start of every trace
3. create a set of all input transitions $T_O = t \in T \mid \exists_{\sigma \in W} t = \mathit{last}(\sigma)$ for every event at the conclusion of a case
4. discover places in the WF-Net by creating a pair tuples of sets $(A, B) \mid A \subseteq T_W \wedge B \subseteq T_W$ to create a set of all transitions $X_W$ where $\forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a1,a2 \in A} a_1 \#_W a_2 \wedge \forall_{b1,b2 \in B} b_1 \#_W b_2$
5. filter $X_W$ for any sets of (A, B) that are non-maximal to set inclusion of transitions creating a new set of transitions $Y_W$

6. a set of places $P_W$ for the output process model are created using the transitions from $Y_W$, including the source place $i_W$ and the output place $o_W$

7. the set of places $P_W$ are connected to their respective input and output transitions from $T_I$ and $T_O$ to establish the set of arcs $F_W$ from $i_W$ and towards $o_W$

8. the output Petri-Net $\alpha(W) = (P_W, T_W, F_W)$ is returned

Given the defined case sequences defined above, once the $\alpha$ algorithm is complete, the resulting WF-Net is produced and illustrated in Figure 3.1.

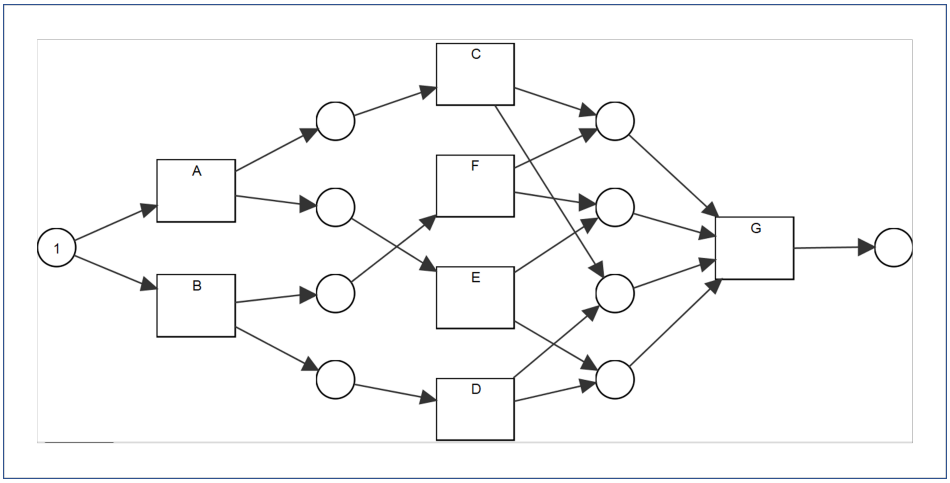



FIGURE 3.1: Example of a Petri-Net

Though the $\alpha$ algorithm outputs process models that are well-defined and useful for gleaning a contextual oversight of a process, it is observed that there are several limitations to using this algorithm. The inherent process in an event log may exhibit setbacks such as tasks that never become active which result in deadlocks in the process. The discovered WF-Net could also contain implicit places that do not affect the overall behaviour of the process. A basic $\alpha$ algorithm also would not supports repetition of events within an event log, so loops would not be shown in a basic WF-Net. Though, it is possible that the algorithm can be improved to discover short repetitive loops, the work by [23] highlights that this is not as trivial as it sounds. Two separate WF-nets can have the same sequence of operators between events in loops but can behave differently.

### 3.2.3 Inductive Miner Algorithm with a Cyber-Attack Scenario

Here, another process mining algorithm is presented called the Inductive Miner (IM) [75] and using an open-source data set of Internet Control Message Protocol (ICMP) alerts [76] generated by a sequence of attack steps to carry out a DDoS attack performed by a novice attacker, the IM is applied to return a process model that illustrates the resulting workflow of the DDoS Attack. The ICMP alerts that comprise said attack are listed in Table 3.2. The discovery of the attack workflow is possible through the use of alert correlation, defined as the process of managing and analysing the numerous alerts generated by Intrusion Detection Systems (IDS) for the purpose of discovering the process-oriented strategies behind cyber-attacks [77].

In ProM, the IM plugin outputs a sound block-structured process model called a *process tree*. A process tree is a model with a hierarchical tree-like structure where the outer leaf nodes represent the event activities and the inner nodes represent the operators such as the sequence or parallelism compositions that define how these nodes relate to each other [78]. A process tree is advantageous in that it handles infrequent behaviour and supports very large data models while ensuring model soundness. The model also supports operators including redo loop sequences for cases with one or more event sequences occurring repeatedly and an exclusive choice between an operator's children.

Given an event log of a finite set of events $L \subseteq \mathcal{L}$, $\sqcup \notin L$ where $\sqcup$ is a silent activity, the IM operates in the following steps:

- identifying the root operator for $L$
- recursively dividing the $L$ into disjoint subsets and assigns a corresponding operator to a characteristic if the characteristic matches the subset
- the recursive division continues through each subset until a single activity is found by itself or with silent activities that cannot be observed and have no impact on the model
- the process model is discovered

The event log of ICMP IDS alerts used to return this process model constitutes a DDoS attack executed by a novice hacker to hack into a variety of hosts around the Internet

[76]. For this activity, an ICMP alert type corresponds to an event in terms of process mining. The stages of the attack and their respective attacks are described as follows:

1. the cyber adversary probes the target network from a remote site to learn which host IP addresses are active
2. the attacker examines a live IP address to see if the vulnerable Sadmind daemon is running which they can exploit sending off several pings the ICMP alert *Admind_Ping*
3. the attacker attempts to hack into the host IP address with the detected Solaris Sadmind vulnerability repeatedly each with a different parameter as part of a remote buffer-overflow-attack to exploit the vulnerable machine, consisting of *Sadmind_Amslverify_Overflow* and *Admind* alerts
4. the attacker installs a DoS trojan on the compromised host using telnet or RCP protocols and generates *Rsh* and *Mstream_Zombie* alerts
5. the attacker launches a single mainstream DDoS attack on the offsite server pinging the *Stream DoS* alert which is only sent once

This data is prepared for process discovery by removing any cases with only one ICMP alert. The event log is then filtered further by removing any alerts that are not part of the desired multi-stage cyber-attack model according to [79] and [80]. The resulting event log only contains 41 alerts and a vocabulary of 6 alert types. A frequency distribution for these alert types for this newly compiled event log is illustrated in Table 3.2:

| ICMP alert | Frequency |
|---|---|
| Sadmind_Ping | 3 |
| Admind | 17 |
| Sadmind_Amslverify_Overflow | 14 |
| Rsh | 4 |
| Mstream_Zombie | 2 |
| Stream_DoS | 1 |

TABLE 3.2: Frequency Distribution of LLDOS Data Set

As shown above in Table 3.2, *Sadmind_Amslverify_Overflow* and *Admind* are the most frequent ICMP alerts in the attack scenario as both are prerequisites to each other and represent the buffer_overflow attack [79]. It is also noteworthy that the Stream_DoS alert

pings only once as the attacker only requires a single stream to launch the DDoS attack. The cyber-attack scenario is illustrated by the process tree illustrated in Figure 3.2.

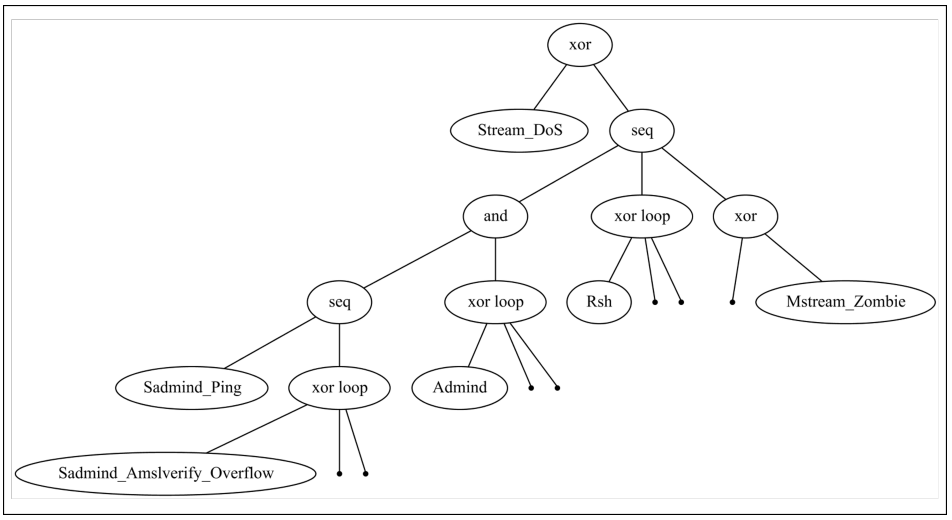



FIGURE 3.2: Example of a Process Tree

As shown in Figure 3.2, there are two available process streams that comply with the process tree. The two available operands include the sequence of probing, exploitation and installation stages of the DDoS attack and the actual launching of the attack. This is due to the DDoS attack only requiring a single separate stream to launch where the source IP address is spoofed to hide the attacker's identity. Another observation is that there is an XOR loop operator for both the *Sadmind_Amslverify_Overflow* and *Admind* representing the buffer_overflow attack used to hack into the sadmind service as the XOR loop allows one of its operands to execute at least once. It is also worth noting the empty nodes appearing in conjunction with the XOR loop operators. These empty nodes represent silent activities whose transitions signify the start or end of a "redo" loop composition.

### 3.2.4 Results & Findings

Having employed the $\alpha$ and the IM algorithms, process models for providing a contextual oversight of an existing general process and illustrating the process-oriented anatomy of an available cyber-attack respectively were generated. These process models would provide cyber personnel with a contextual oversight of the two processes which would both be useful for cyber-risk assessment which improves cyber situational awareness.

While cyber-security personnel would appreciate the illustrative contextual oversight gleaned from these process threads, these process mining tools lack any actual means of evaluating the effectiveness of modelling the overall process threads. Using these process models, there are no quantifiable means of performing process flow prediction. Because of this drawback, the objective for gaining contextual oversight of a general application process shifted from the application of process mining to training a deep learning model to discover the contextual oversight and make predictions in ongoing threads.

## 3.3 Deep Learning Approach

Another method to discovering a contextual oversight of a general application process to improve cyber situational awareness is to train a deep learning (DL) model to learn the typical form of a general application process.

The model discovers existing sequential dependencies within the general process threads, or cases, to discover how one event type at a particular time step could impact the remainder of a single case, thus gaining a contextual oversight of the general process. Using two different methodologies, the DL model will be trained and its performance will be evaluated based on its ability to make predictions about ongoing cases, similar to the work by [29]. These two methodologies will then be compared to in terms of prediction accuracy to evaluate their effectiveness in improving cyber situational awareness. This approach was published in [81].

In this section, the objective, design and parameters of the DL model are presented, four open-source data sets that will be fed into the model as input are described, how the data is prepared for the DL model is outlined and the two techniques used to process the data and train the model are described.

### 3.3.1 Objective of the Deep Learning Model

The aim is for the DL model to learn existing prefixes of an input sequence and predict the corresponding suffix in ongoing process executions. RNNs are a class of ANNs that feed the output values back into the hidden layer(s) over a number of time steps as illustrated in Figure 3.3. In this way, long-term dependencies within the data can be

relearned and updated regularly, which affects the likelihood of probable events occurring later in the case. Therefore, an RNN is ideal for modelling sequences of events and learning the inherent dependencies.

After the RNN model predicts a future event in the ongoing case, the predicted value will be compared to the actual value at the current time step of the case. A loss function will be used to calculate the difference between the ground-truth value and the predicted value. This loss value will be minimized using an algorithm called Back-propagation through time (BPTT).

### 3.3.2 LSTM Model Architecture Design

The primary building block used for the RNN model is a Long Short Term Memory (LSTM) network developed using the software library Keras [82]. The execution paths of processes can exhibit long or short term dependencies. LSTMs are used to model noisy, sequential data, and can discover and memorize these existing dependencies in the data and hence are suited for modelling the sequential form of the event logs. The output layer for the RNN will be a Dense layer where every node of the layer is connected to all the nodes of the previous layer in the network. The LSTM model architecture described is shown in Figure 3.3.

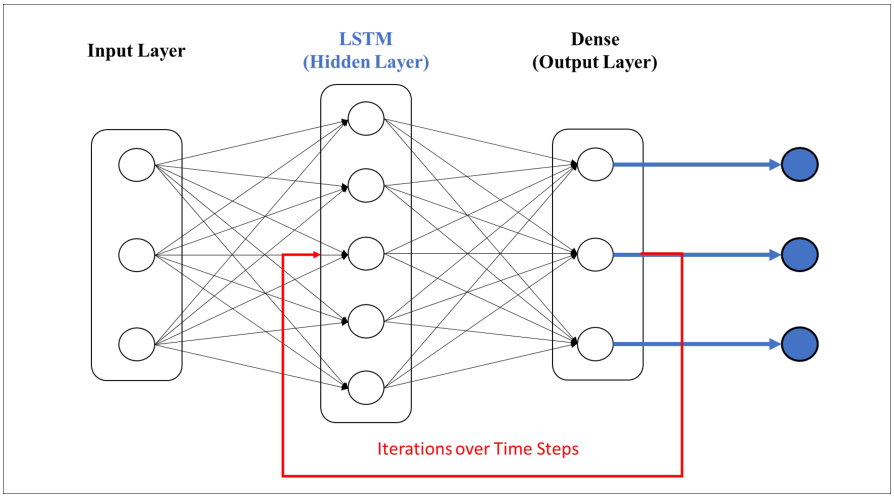



FIGURE 3.3: LSTM/RNN Model

Given the multiple unique event types contained in the event logs, the deep learning problem is a case of classification. In the Dense layer of the LSTM architecture, each node will correspond to a different event class, and a softmax activation function is used

to define the nodes in the output layer. This returns a probability distribution vector over the number of different classes. Finally, being a non-binary classification problem, a *categorical_crossentropy* loss function is used during the compilation of the model.

Normally, the size of the RNN network is that of a probability distribution over all unique event types for every sequence. Therefore, the illustration in Figure 3.3 represents the primary build of the RNN model to be used. However, in an alternate case, it is possible to configure the RNN to return a probability distribution, or a sequence of predictions, at every time step for every sequence. Using the Keras library, this can be achieved by embedding the Dense layer in a TimeDistributed wrapper layer. The end result is a prediction of event types at every time slice within a sequence. This alternate design was used in the teacher forcing method.

### 3.3.3 Data Sets

For carrying out this approach, four available data sets are introduced and listed as follows: the Business Process Intelligence Challenges (BPIC) for 2012, 2013, 2014, and the Helpdesk data set used to evaluate the approach by [27]. These data sets are outlined in detail below.

#### 3.3.3.1 Helpdesk Data Set

This data set is an event log from a ticket management process for the help desk for an Italian software company. This was used as supplementary material for [27]. This event log was composed of 13710 events, 3804 cases and had a vocabulary of 9 different event types. Each event had their own *Activity ID* and a single *Case ID* being the unique case identifier.

#### 3.3.3.2 BPIC 2012

This data set is an event log used for the BPIC workshop in 2012 [83]. This is a log for an application procedure for financial services at a Dutch financial institution. This event log was originally comprised of several sub-processes, but like the works by [27] and [84], the log was narrowed down so that only cases with the *work item* process that

began with event types *start* and *complete* were included. The resulting event log was comprised of 7469 cases and had a vocabulary of 6 event types. Each event had its *ActivityID* in the data and were identified and grouped by their *CaseID*.

#### 3.3.3.3 BPIC 2013

The BPIC data set for its workshop in 2013 [85] is an event log for VINST, an incident management system that solves IT related incidents for the Volvo Information Technology company in Belgium. Request calls sent to VINST are treated as a case with a unique identifier being the *Service Request* number. This data set was comprised of 7553 cases. For this work, both the works by [81] and [86], a vocabulary of 13 event types were generated by combining every possible combination of the two columns *Status* and *Sub Status*. For promoting the simplicity of this project, each event type is mapped to a corresponding event number. Each case of events will be grouped using the the*SR NUmber* as a unique identifier.

#### 3.3.3.4 BPIC 2014

This data set was compiled from a collection of three different processes performed by the ICT Department of a financial services company called Rabobank Group from the BPIC for 2014 [87]. A customer will send request calls which are logged by a service management tool. Three main sub-processes which are described as follows:

- *Interactions*: Rabobank employees send request calls to the ICT department's service desk where a Service Desk Agent (SDA) answers the call, resolves the issue and logs the call as an *Interaction*
- *Incidents*: if the SDA is unable to resolve the technical issue, the problem is assigned to an Assignment Group and the process the assignment group takes to resolve the technical disruption is logged as an *Incident*; each incident is treated as a case of logged process events
- *Changes*: if the disruption were to occur more than once, an investigation is launched to analyse the problem to perform an improvement plan to ensure the

problem never happens again, and the procedure to implement this plan is logged as *Change*

The primary Rabobank sub-process investigated for this thesis is the *Incident* data. The actual event log used to evaluate the deep learning approach is a prepared data set built using the *.csv* files recording the data regarding every incident. The process to prepare this new data set is described as follows. The files downloaded from the workshop for this process are *Detail Incident.csv*, a list of 46605 logged incidents and *Detail Incident Activity*, an event log of 466739 related events each belonging to a single incident. For each incident, the attribute *IncidentID* is the designated identifier. For this work, the vocabulary of different event types was defined using the column *Category* from the incident log and *IncidentActivity-Type* from the incident activity log. First, the two *.csv* files were then merged into a new event file using the *IncidentID* column shared between the two files as a joining key. Second, every possible unique combination of the two aforementioned columns *Category* and *IncidentActivity-Type* is generated to create a vocabulary of 91 event types. Finally, the each incident of related event types is grouped by their shared *IncidentID* returning a new data set of incidents for this work. For simplicity, a small subset of the whole data set composed of only 6000 cases with a vocabulary of 69 event types is produced and trained.

### 3.3.4 Data Preparation

In order to train the LSTM model defined in Section 3.3.2 to discover the process contextual oversight for each of the four data sets, the logged data for each data set is prepared by grouping each event by its respective case in chronological order. The data is converted into a 3-dimensional array representation (sequences, time steps, event types). The first dimension represents the number of sequences to train. The second array is the number of time steps within a chronological ordered case. Finally, the length of the third dimension is equal to the number of event types present in the event log's vocabulary and the event at the specific time stamp is one-hot encoded. The layout for this input data matrix is displayed in Figure 3.4. The data representation for the largest data set, the full BPIC 2014 data set with 466739 events, was prepared in approximately 9 seconds.

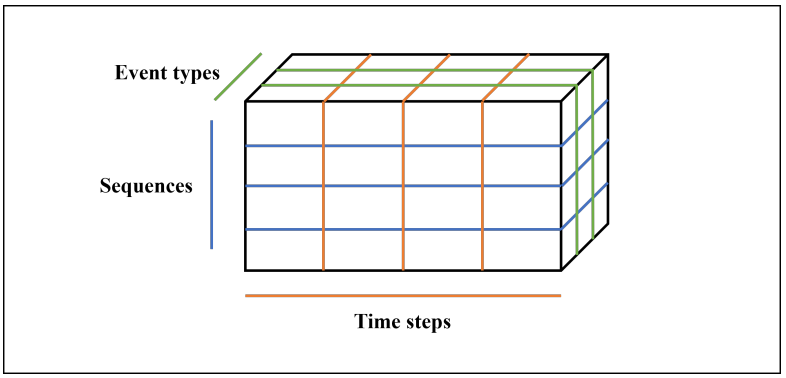



FIGURE 3.4: Input Data Array for the LSTM Model

### 3.3.5 Methodologies for Training an LSTM Model

In this section, the two proposed methodologies to train the LSTM model, defined in Section 3.3.2 for learning the sequential form of cases and predicting subsequent events to gain the contextual oversight of the data set are described. These two methods are the **prefix method** [27] and the existing **teacher forcing** methodology outlined by [88].

#### 3.3.5.1 Prefix Methodology

This methodology, used by [27], takes a set of cases and generates a new set of prefixes with a length longer than a single event for every case. Each prefix will be used to predict a single subsequent event in their original sequence. For example, given a case with the sequence 1, 2, 3, 4, 5, 6, the resulting input prefixes defined as **X** and their corresponding output suffixes defined as **y** are listed in Table 3.3.

TABLE 3.3: Input and Output for the Prefix Approach.

| **X (input)** | ***y (output)*** |
|---|---|
| [1, 2] | 3 |
| [1, 2, 3] | 4 |
| [1, 2, 3, 4] | 5 |
| [1, 2, 3, 4, 5] | 6 |
| [1, 2, 3, 4, 5, 6] | *!* |

The resulting set of prefixes will be used to train the LSTM. Therefore, the model was essentially trained to learn the form of the input prefix **X** and used to predict the output suffix **y**. While implementing this methodology, it should be noted that the LSTM model

will also predict a *!* character which represents the actual completion of a case. It is also notable that for every case, the shortest prefix is of length two since it is not really feasible to detect sequential dependencies for a case with only a single event type.

#### 3.3.5.2 Teacher Forcing Methodology

Using the teacher forcing method [70], [88] a subsequent event can be predicted by training the ground truth value at the previous time step as input, and the process is repeated along the sequence in question. For example, given the previous sequence 1, 2, 3, 4 5, 6, the second event **2** is predicted given input **1**, and so on. The existing input and output of the sequence for this methodology is displayed in Table 3.4.

Table 3.4. Input and Output for the Teacher Forcing Methodology.

| ***X (input)*** | ***y (target)*** |
|---|---|
| [1, 2, 3, 4, 5, 6] | [2, 3, 4, 5, 6, !] |

During the training of the LSTM model, the entire sequence is iterated through the model only once. Notice that in Table 3.4, the last index of input **X** and the first of **y** are removed. This model takes the ground truth value of the index preceding the current time step of the sequence.

In order to train a data set of cases using this methodology, the LSTM model defined in Figure 3.3 will be implemented but with the Dense output layer being wrapped in TimeDistributed wrapper layer. In order to implement the teacher forcing methodology, the model must be able to predict a subsequent event at every time step. Using the TimeDistributed wrapper, a Dense output layer can be distributed at every time slice of the input data sequence during the training process.

### 3.3.6 Training & Evaluation of LSTM Model

Here, the training process and evaluation for both of the LSTM-based methodologies defined in Section 3.3.5 are outlined. Several different hyper-parameters used for training the model are also presented. The source code and test beds demonstrating the two different LSTM models for all four data sets are stored in the following repositories as follows: BPIC 2012 [89], 2013 [90], 2014 [91] and the Helpdesk [92].

#### 3.3.6.1 Prefix Method

For the training data using the prefix method, every possible prefix is generated using the approach outlined in Section 3.3.5.1. To define the prefix data along the second dimension of the input data matrix, all prefixes are pre-padded with zeros to the length of the longest existing case. This makes the defining of the one-hot encoding for the vocabulary of event types along the third dimension of the input array. When training the model, additional parameters include the number of epochs and the batch size. Finally, 20% of the training data is set aside for validation purposes to evaluate the model's performance. Advantages of this validation include observing the behaviour of the accuracy/loss values at the completion of every epoch.

When evaluating the trained model with the prefix methodology, all possible prefixes for the testing data set are likewise generated. Each testing prefix is fed into the network, and the responding output is a probability distribution over all unique event types. Every distribution array of event types is set using a predefined sequence of event types. For the model to determine the predicted subsequent event, the index of the largest variable in the probability vector is indexed and its respective event is returned by the model. The predicted subsequent event is compared to the actual event type for every testing prefix to determine the model's prediction accuracy.

#### 3.3.6.2 Teacher Forcing Method

For the Teacher Forcing methodology, the training cases is sorted by increasing size and the data is divided into mini-batches of a specified size. The model will then train on each of these mini-batches. All batches must be of the same length, so all cases are pre-padded to the length of the longest case in their respective mini-batch.

To test the model, every sequence is processed by the LSTM network and the output is a sequence of probability distributions at every time step in the sequence. The resulting event types at every time step of the resulting sequence belongs to the index of the largest value at their respective distribution vectors. Accuracy, in regards to the model's performance, is calculated by comparing the predicted event with the ground-truth subsequent event in the input case starting after the second event.

### 3.3.7 Results & Findings

Having built the LSTM network proposed in Section 3.3.2, the model was trained using a range of several parameter values to discover the optimum configuration of meta-parameters and thereby the best possible performance by the model. This was explored by training the model with the BPIC 2012 data set with the Prefix methodology which is illustrated in Table 3.5.

For both methods, the LSTM model was trained using an Adam optimizer [93], used for sparse gradient descent while training deep neural networks with noisy data. The Adam optimizer is efficient due to requiring minimal memory and parameter tuning. The optimizer also uses an adaptive learning rate [94], a hyper-parameter used for computing different network weights for maintaining the step size at every iteration while training the model. This provides a trade-off between providing the solution in a timely manner and overshooting the optimal solution.

Table 3.5. Prefix Methodology with the BPIC 2012 Data Set.

| LSTM | Dropout | Nodes | Batch Size | Epochs | **Accuracy** |
|---|---|---|---|---|---|
| 1 | 0 | 100 | 10 | 50 | 65.68% |
| 1 | 1 | 100 | 10 | 50 | 66.31% |
| 2 | 0 | 100 | 10 | 20 | 66.34% |
| 1 | 0 | 100 | 6 | 20 | 66.60% |
| 1 | 0 | 120 | 32 | 20 | 67.73% |
| 1 | 0 | 60 | 32 | 20 | 67.88% |
| **1** | **0** | **100** | **32** | **20** | **68.64%** |

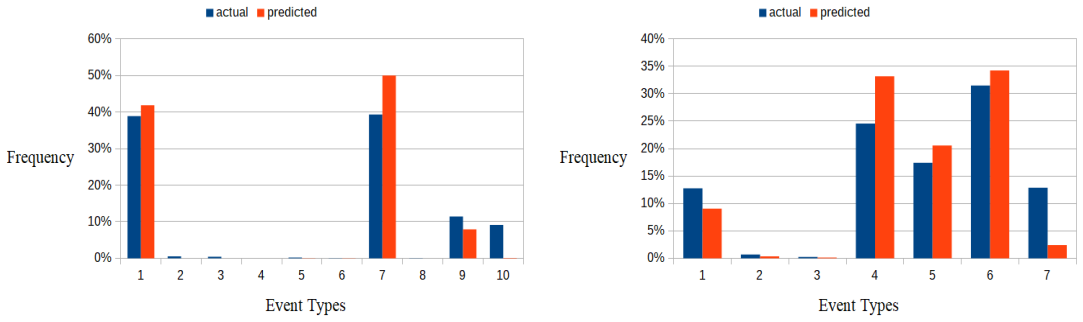
The maximum accuracy values for correctly predicting a subsequent event in each data set are listed in Table 3.6.

Table 3.6. Maximum Accuracy Values for all Data Sets.

| **Data Set** | Cases | Events | **Max Acc.** |
|---|---|---|---|
| **Helpdesk** | 3803 | 9 | 81.31% |
| **BPIC 2012** | 7469 | 6 | 68.64% |
| **BPIC 2013** | 7553 | 13 | 65.66% |
| **BPIC 2014** | 6000 | 69 | 48.28% |

As shown in Table 3.6, the Helpdesk data set has the highest accuracy of approx 80%. It is also inferred that the bigger the vocabulary of event types in the data sets, the harder

it is to make predictions about the data. The frequency distributions over the event types in the Helpdesk and BPIC 2012 data sets were calculated and both the actual and predicted data are highlighted. These observations are displayed in Figure 3.5. The vocabulary of events for both data sets will include a *!* character that signifies the completion of a case.




(a) Event Types for Helpdesk Set Set

(b) Event Types for the BPIC 2012 Set

FIGURE 3.5. Frequency Distributions of Data Sets

As displayed in Figure 3.5(a), five of the event types from the Helpdesk data set were shown to be rather infrequent in the testing data. The BPIC data set has a smaller vocabulary but sports much more frequent data than that of Helpdesk as shown in Figure 3.5(b). Notice that infrequent events tend to be under predicted while more frequent event types tend to be over predicted in both data sets. Because the BPIC 2012 has a more frequent spread of event types, both actual and predicted, this could explain why the Helpdesk data set returns a greater accuracy than the BPIC data set.

It is worth noting that a similar situation is true for the 2013 data set, where out of the 13 existing event types in the vocabulary, only five are shown to be rather infrequent. In Figure 3.6, the frequency distributions for all event types for the actual testing event data and the two predicted values returned by the two methodologies are displayed. It is also observed that the Teacher Forcing method is more prone to this bias than the Prefix method.

Table 3.7 displays the resulting accuracy values and execution times for all four data sets using both training methods.

From Table 3.7, it is observed that the Teacher Forcing method trains much faster that the prefix method. As illustrated by Table 3.3, the prefix methodology is shown, for a case of length $n$, to generate $n$ - *2* prefixes. Consequently, the training instances for the

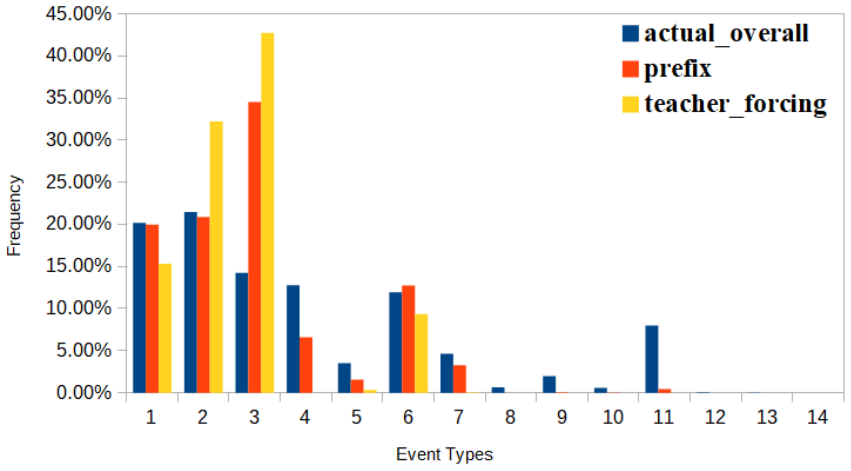



FIGURE 3.6. Frequency Distribution of Test Events for BPIC 2013

TABLE 3.7. PREFIX AND TEACHER FORCING COMPARISON.

| **Data Set** | **Cases** | **Events** | **Prefix Method** | | **Teacher Forcing Method** | |
|---|---|---|---|---|---|---|
| | | | Accuracy | Time (mins) | Accuracy | Time (mins) |
| Helpdesk | 3803 | 9 | 80.39% | 0.97 | 81.31% | 0.17 |
| BPIC 2012 | 7469 | 6 | 68.64% | 17.9 | 68.19% | 0.6 |
| BPIC 2013 | 7553 | 13 | 65.66% | 32.1 | 63.31% | 0.63 |
| BPIC 2014 | 6000 | 69 | 48.28% | 42.5 | 43.68% | 0.5 |

prefix method is shown to be $n$ times larger which accounts for the substantial difference in training times. Furthermore, it is shown that both techniques produce similar results for both the Helpdesk and BPIC 2012 data sets. Surprisingly, despite using the same loss function and hyper-parameters, the prefix method outperforms the teacher forcing method in the case of the other two methods though only by a small margin.

The full BPIC 2014 data set contains 46606 cases. Training with this new data was performed on an NVIDIA GPU server with a four-card Tesla SXM2. For the Prefix method, the results of training and making predictions on the full data set include a prediction accuracy of 49.49% and a training time of one hour and 7 minutes. Given a vocabulary of 91 event types, the end result is quite satisfactory as a random guess would result in an accuracy of approximately 1%.

These results and determinations for this proposed deep learning approach are published in [81].

## 3.4 Conclusion

Due to the immense number of alerts generated by intrusion detection software, cyber-security personnel struggle to attend to these alerts which hinders their ability to pre-empt cyber-attacks targeting their software enterprises. One approach for personnel to enhance the prevention of cyber-security attacks is to improve their cyber situational awareness through application process flow prediction. This can be achieved by gaining a contextual oversight of their general application process and make subsequently predictions about future events in ongoing processes. This helps personnel to identify assets and services of the organisation being targeted, prioritise the more severe cyber-attack events and allocate appropriate resources to address the impending cyber-threat.

First, the application of process mining was investigated to produce an outright process model given an input event log. Two process mining algorithms that can be implemented as plugins for a process mining tool to output process models were studied. it was concluded that the first PM algorithm, the $\alpha$ algorithm, was effective but limited in that the resulting process models could only capture a process workflow perspective of data and not frequencies of events and repetitive patterns in the process model. The second algorithm explored, the IM, proved to support additional ordering relations between logged events including exclusive choice and redo loops which the first lacked.

Though the results from the $\alpha$ and IM process mining algorithms were useful for displaying an illustrating overview for an existing process, this proposed approach was limited because there are no quantifiable means to evaluate the performance of the model. For this reason, the primary objective for this part of the project shifted towards the deep learning approach.

The application of deep learning models, specifically LSTMs, were investigated for gaining a contextual oversight of a general application process. The LSTM model in question was trained to learn the dependencies of a sequential case of events and subsequently used for predicting future events given an input prefix of a test case. To evaluate this approach, two methodologies of training an LSTM were applied and tested using four available data sets. The accuracy of predicting subsequent events depends heavily on the number of event types found in a data set. It is far easier to make predictions for data sets with a smaller vocabulary of event types. It was also observed that trained models

tend to over predict more frequent events and under predict infrequent events. This bias was found to be more pronounced using the teacher forcing method. To evaluate the LSTM's effectiveness in gaining a contextual oversight of an application process, these two methods were compared to each other in terms of prediction accuracy.

It is noteworthy that the Helpdesk data set had the highest accuracy when predicting future events at 81% with only 9 event types. This is high, compared to the BPIC 2012 data set which has a vocabulary of only 6 event types yet has a resulting accuracy of 68.64%. However, five event types from the Helpdesk data set were shown to have a low frequency whilst testing resulting in the model being more biased towards the remaining event types which could explain the higher accuracy. Furthermore, for two of the data sets, Helpdesk and BPIC 2012, the prediction accuracy was shown to have very similar results, being less than 1% difference. In the other two data sets, the prefix was proven to produce slightly higher results than the teacher forcing method, with a 2.3% difference for the BPIC 2013 data set and 4.6% for the BPIC 2014 data set.

By being able to perform application flow prediction based on an oversight of their general application process, cyber-security personnel would greatly benefit from using these two LSTM model-based approaches as they would gain an improved cyber situational awareness of their software enterprises. Advantages of this approach include being able to identify potential targets in ongoing threads of their system application and being able to pre-empt impending cyber-threats. Also, by comparing the LSTM based models, it is determines that the teacher forcing method takes an appreciably shorter time to train than the prefix method, by a factor of up to six times faster. Therefore, by using the teacher forcing LSTM method in particular, cyber-defenders will be able to quickly address cyber-threats to their system applications which supports the prevention of cyber-attacks.

# Chapter 4

# Anomaly Detection by Frequency Distribution of Microservices Application Tracing

## 4.1 Overview

Over the past decade, the microservices architecture (MSA) has become immensely popular because if its loosely coupled design, increased scalability, and support for agile software team development and automated deployment over a variety of software environments. The MSA is now the leading method for application development and has been adopted by many commercial software enterprises.

However, the architectural design increases the attack surface due to the overall application being composed of several interconnected services which must operate alongside each other to function. The exploitation of a single service may propagate throughout the application, leading to all the application's trusted peers, providers and services being compromised [95]. Due to many recent attacks on various companies which use the microservices architecture like Amazon and Netflix in recent years [96], enhancing the detection of cyber-attacks targeting microservices applications is an urgent need nowadays. Because of the distributed design of microservice activity it is rather difficult to monitor and navigate microservices and detect cyber-attacks targeting these applications.

In this chapter, the field of microservices, and the topic of frequency distribution are introduced, the capability to monitor microservice functionality using distributed tracing is described and explore whether anomaly detection techniques can be applied to microservice application traces. For this contribution, the objective is to apply anomaly detection techniques to microservice application traces to detect injected cyber-security attacks. It is proposed to use frequency distribution-based anomaly detection to prove that injected attacks in a microservices application result in an irregular quantity of specific functionality. For this contribution, an open-source microservices application *SocialNetwork* [97] from an available test-bed called *DeathStarBench* [98] is run. Based on the methodology described above, the following research question is addressed.

*Can frequency distribution-based anomaly detection be used to detect cyber-security attacks in a microservices application?*

## 4.2 Frequency Distribution

The term frequency distribution is defined as an overview of distributed values for a single random variable and the corresponding frequency for each value that occurred within a certain volume of space or time [99]. For example, in a university, a study is conducted concerning a select set of subjects to learn the number of university students enrolled in each subject as illustrated in Table 4.1.

| **Subject** | **No. of Students** | **%** |
|---|---|---|
| Psychology | 28 | 15.56% |
| Computer Science | 28 | 15.56% |
| Chemistry | 32 | 17.78% |
| Physiology | 31 | 17.22% |
| Biology | 32 | 17.78% |
| Sociology | 29 | 16.11% |
| **Total** | **180** | 100% |

TABLE 4.1. Frequency Distribution Example

Table 4.1 shows the frequency distribution of the number of students over the six subject courses listed. The relative frequencies of students compared to the total number of students listed in the study is also shown as proportions or percentages. From these relative distributions insightful information about the frequency distribution can be gained. From this illustration, it is inferred that the subjects 'Chemistry' and 'Biology'

are equally the most popular subjects taken both containing 32 students, the highest number in the study.

## 4.3 Experiment

### 4.3.1 DeathStarBench

In this section, the DeathStarBench [98] benchmark suite is outlined, a high-level description for one of the available microservices-based applications is provided, including its main properties, components and available microservice functionality. DeathStarBench is an open-source benchmark suite of microservices with a focus on large scale applications composed of tens of microservices. Besides *SocialNetwork*, the available applications that compose the benchmark suite include a movie review application, a banking system, an e-commerce website and a hotel reservation application.

#### 4.3.1.1 *SocialNetwork*

The *SocialNetwork* application [97] from the DeathStarBench test-bed emulates a broadcast-style social networking application composed of uni-follow relationships between users. The architecture of this application is displayed in Figure 4.1. The application itself receives users' HTTP API requests for requested microservice functionality in the frontend client component. These API requests are initially sent to a web browser/load balancer component implemented using NginX [100] which delegates the requests to the appropriate microservices. Various client API requests that are sent to the application logic are listed as follows:

- *api/user/login*
- *wrk2-api/user/register*
- *wrk2-api/post/compose*
- *wrk2-api/user/follow*
- *wrk2-api/user-timeline/read*

These downstream API calls or Remote Procedure Calls (RPC) also interact with other neighbouring microservices as requests can span across multiple service components. These RPCs are implemented using the Thrift [101] framework.

On the server side of the application, data storage is provided by a MongoDb service and caching of the data is handled by a Redis and Memcached as illustrated in Figure 4.1. For *SocialNetwork*, the distributed calls to the microservice operations are recorded by the **Jaeger Agent** daemon, stored in the **Jaeger Collector** which receives queries from the **Jaeger Query** component. The collector component also leverages the **ElasticSearch** API and the Kibana interface for storing and observing the microservice data as JSON documents.

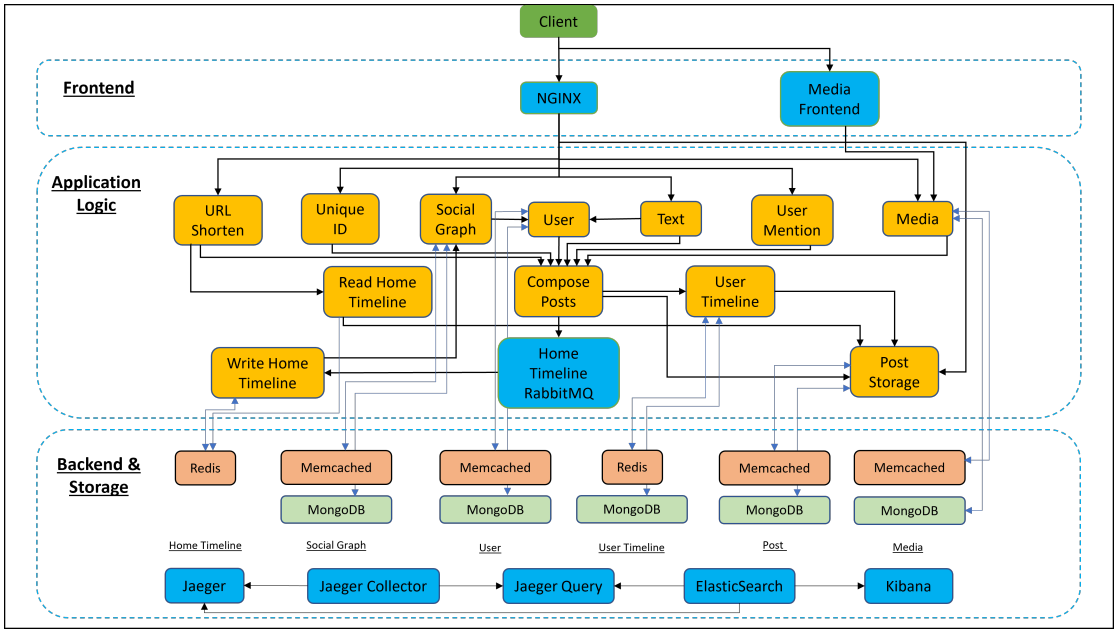



FIGURE 4.1. Microservices Application Architecture of **SocialNetwork**

### 4.3.2 Software & Hardware Environment for Experiment

Here, the software environment where the application *SocialNetwork* was executed and the software tools which the application was configured to use are outlined. The application itself was run on an Intel Core i3-2370 CPU processor. The programming languages used to hard-code the application include Python 3.7, Java, JavaScript, node.js, lua, PHP, C and C++. The API request calls made to the application were sent using scripts written in Python 3.7, and the AI and machine learning libraries Tensorflow 1.13 and Numpy 0.19.0 were implemented.

#### 4.3.2.1 Docker

Docker is a Platform-as-a-Service (PaaS) tool used for developing, deploying and running applications in a virtual environment. Docker is used to compress all existing source code, dependencies and libraries into isolated and immutable files called Docker *images*. An image can be shared as open-source by their developers and used as a template for executable source files called Docker *containers* which software developers can run and modify at their leisure. In the *SocialNetwork* application, every individual microservice is run as a Docker container in coordination with a tool called *docker_compose*. The function of *docker_compose* is to create, configure and run the existing microservices to be utilized in the application.

#### 4.3.2.2 Thrift

Thrift is an interface definition language and binary communication protocol developed by the Apache software Foundation [101]. Thrift is used to define data object types, their attributes and service interfaces using a definition file. This definition file supports client-server RPCs. The Thrift language supports cross-language service development across several languages which works amongst several programming languages including Java, C++, Python, PHP, Ruby, Node.js and C#. In *SocialNetwork*, every RPC sent to a microservice is provided by Thrift, and the interfaces for these microservices are defined in a *.thrift* file written in the Thrift language.

#### 4.3.2.3 Jaeger

For distributed tracing the application was configured to use an open-source distributed tracing system called Jaeger [102]. The primary function of Jaeger is to provide performance bottlenecks for distributed systems with a microservices architecture. By configuring an application with the Jaeger distributed tracing system, the distributed functionality in a running application can be tracked and monitored for performance analysis. Jaeger is used for several troubleshooting activities which include distributed transaction monitoring and context propagation, root cause analysis, and performance latency and optimization.

The Jaeger system, as a whole, can be described through the following components:

- **Jaeger client**: a language specific implementation of a distributed tracing service. This component is used to instrument applications for distributed tracing, used for creating spans in response to outgoing request API calls, attaching context information to the span and sending the span to the Agent component

- **Jaeger agent**: a network daemon that listens for actual executing spans sent by the client services over a UDP connection. The agent abstracts the routing and discovery of the recorded span away from the client. Once the agent receives the span data, it is batched and sent to the Collector component for storage.

- **Jaeger collector**: receive traces from the Agent, validates and indexes the received traces, and finally stores them. The collector supports the various search APIs for storing document objects such as ElasticSearch and Cassandra.

- **Jaeger Query**: a service used to query for and retrieve trace data stored in the Collector. Once a logged distributed trace or event span is selected, a host UI is employed to display and analyse the data.

For this contribution, the *SocialNetwork* application was configured with the following Jaeger components: the **Jaeger agent**, the **Jaeger collector** and the **Jaeger query**. The collector component will also be configured to use the ElasticSearch API [103] as a storage backend. This stores the logged span data in the form of JSON documents. Using the ElasticSearch API, it was made possible to download the data in bulk. This implementation of Jaeger is illustrated in Figure 4.2. The source code for this implementation of the *SocialNetwork* application is available at [104].

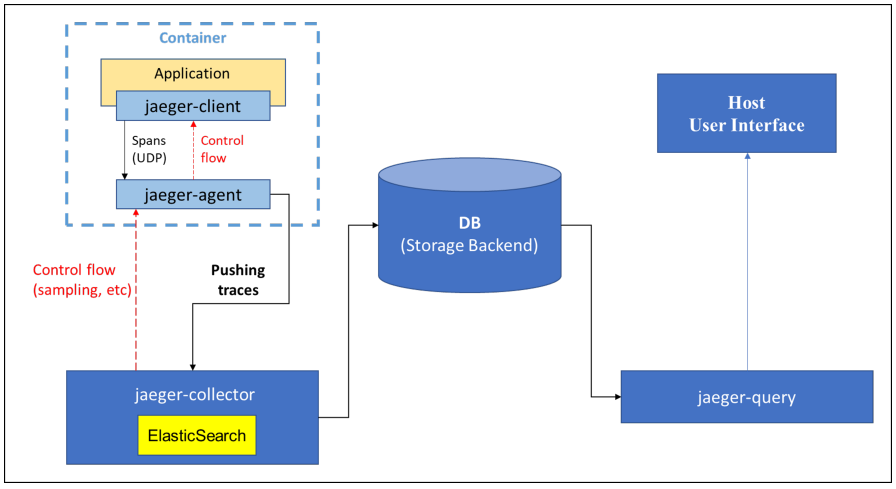



FIGURE 4.2. Jaeger Implementation of the Application Architecture

## 4.4 Results & Findings

In this section, two well-known cyber-attacks are analyzed: a brute force password guessing attack and NoSQL Injection. By running the *SocialNetwork*, it is examined if it is possible to use distributed tracing and anomaly detection to detect injected cyber-attacks by calculating the frequency distribution of distributed traces. The results of this experiment were published in [105], and the source code data and analysis are available at [106].

### 4.4.1 Brute Force Password Guessing Attack

The distributed application *SocialNetwork* supports a microservice with a login operation, so it is feasible to perform a password guessing attack for this experiment. The simulated attack results in multiple login traces within a short period of time. The irregular microservices activity would be classified as a group anomaly. In a normal situation, only one or two login requests would correspond to normal microservice activity. The aim of this experiment is to detect this group anomaly by determining if the frequency distribution of traces for a user logging into their account differs from a base distribution of normal data.

#### 4.4.1.1 Frequency Analysis of Distributed Traces

A user's API request to a microservice application will return a single trace i.e., a sequence of spans as it propagates from microservice to microservice through the application. In this work, each span is represented by a combination of a microservice name and an operation name.

A distance metric is defined between two sets of traces called $T_1$ and $T_2$. Given all API requests are recorded and logged using distributed tracing, let $s_1, s_2 ... s_n$ be the set of all unique traces in both sets. The frequency distributions $f(T_1)$ and $f(T_2)$ are defined in Equation 4.1:

$$f(T_1) = (s_1, f_1^1) + (s_2, f_2^1) + ... + (s_n, f_n^1)$$
$$f(T_2) = (s_1, f_1^2) + (s_2, f_2^2) + ... + (s_n, f_n^2) \tag{4.1}$$

where $f_i^j$ is the frequency value for trace $s_i$ in $T_j$. The difference between the two frequency distributions is defined using the Euclidean distance metric and is defined in Equation 4.2:

$$d(T_1, T_2) = \sqrt{(f_1^1 - f_1^2)^2 + ... + (f_n^1 - f_n^2)^2} \tag{4.2}$$

#### 4.4.1.2 Application User Requests

To carry out the experiment, the *SocialNetwork* application was executed. Users' HTTP API calls were sent to make two different types of service requests: **composePost** where a user composes and uploads a post to the social network consisting of content such as message texts, tags or web links, and **userLogin** when a user logs into an account.

In *SocialNetwork*, after a user's post is uploaded to the application logic, the post is stored in a MongoDB database [107] and cached using a Memcached service [108] as illustrated in Figure 4.1. For the **composePost** request, there are in total 27 different microservice calls executed and 1308 possible microservice traces. The following lists some of the microservice calls executed during a **composePost** request: *UploadMedia*, *UploadText*, *UploadUniqueID*, *UploadURLs*, *UploadUserMentions*, *StorePost*, *MongoInsertPost* and *MmcSetPost*.

When a **userLogin** request is sent, the application checks the caching service if an appropriate user object has ever logged in before. If not, a call is made to the MongoDB database to check if the user has registered with the application and the user's information is returned. If the user's information has been found, a microservice call caches the user credentials in Memcached. For this request, four calls executed are *Login*, *MmcGetLogin*, *MongoFindUser* and *MmcSetLogin*. The following are valid sequences of microservice calls that occur during a **userLogin** request and are listed as follows:.

1. *Login* - *MmcGetLogin*

2. *Login - MmcGetLogin - MongoFindUser*

3. *Login - MmcGetLogin - MongoFindUser - MmcGetLogin*

The first sequence of microservice calls is executed whether the entered password for the **userLogin** request was correct or incorrect. The second sequence of calls only corresponds to an incorrect sequence call as the user's credentials have not been cached to Memcached and the entered credentials aren't stored in MongoDB. The third sequence only corresponds to a correct login because the user's credentials have been found in MongoDB and were subsequently cached in Memcached.

#### 4.4.1.3 Definition of Normal Application Data

For this experiment, normal application traffic is composed of **composePost** and **userLogin** requests with a correct password. There will also be minimal incorrect **userLogin** requests, typically caused by users mistakenly entering their wrong credentials once or twice. A study by [109] stated that approximately 10% of login requests in a system application are incorrect.

In the experiment, a set of user API calls for both types of requests were sent to the application *SocialNetwork*, resulting in a total of 2550 distributed traces with 32 unique span event types. The total number of 1856 **composePost** traces were returned using a HTTP traffic generator. The number of 694 **userLogin** requests were also sent using hard-coded scripts including 592 correct and 102 incorrect login traces. For this experiment, 2000 of these traces were set aside to make up a training data set. The distribution for the normal data made up of the three mentioned types of traces are displayed in Table 4.2.

TABLE 4.2. Normal Data Set

| **User Requests** | Frequency |
|---|---|
| **composePost** | 1526 |
| correct **userLogin** | 420 |
| incorrect **userLogin** | 54 |
| **Total** | 2000 |

#### 4.4.1.4 Definition of Validation Data

A set of validation data was also comprised of normal data. This data set was composed of 500 distributed traces, consisting of 300 **composePost** traces, 165 incorrect and 35 incorrect **userLogin** traces. (Note that this proportion of incorrect login requests is higher than documented by [109], but this makes anomaly detection harder.) For this experiment, the validation data was divided into 10 sub data sets each of 50 distributed traces. Given the distances from each of these subsets to the normal starting data, the objective for these subsets were to calculate the mean and standard deviation of these distances.

#### 4.4.1.5 Injected Cyber-Attack Data

A third data set with the distributed traces caused by an injected brute force or password guess attack simulated against *SocialNetwork*. This data will contain more incorrect **userLogin** traces than correct traces compared to the normal and validation sets. The distributed trace frequency for this anomalous data set is illustrated in Table 4.3.

TABLE 4.3. Anomalous Data Set

| **User Requests** | Frequency |
|---|---|
| **composePost** | 30 |
| correct **userLogin** | 7 |
| incorrect **userLogin** | 13 |
| **Total** | 50 |

#### 4.4.1.6 Experiment & Evaluation

The experiment using the prepared data sets described above is outlined as follows. First, the Equation 4.1 was used to calculate the frequency distribution $f(T_0)$ for all unique traces in the normal data set. Second, the process is repeated for all 10 validation subsets to calculate the data distribution for each: $f(T_1)$, $f(T_2)$ ... $f(T_10)$. In the next step, the distance metric in Equation 4.2 is used to calculate the difference from the normal frequency distribution to each of the validation distribution data sets. These differences in the data distributions are illustrated in Figure 4.3.

In the final step of this experiment, using Equation 4.1, the frequency distribution for the data set with the injected password guess attack denoted as $f(T_11)$ is calculated.

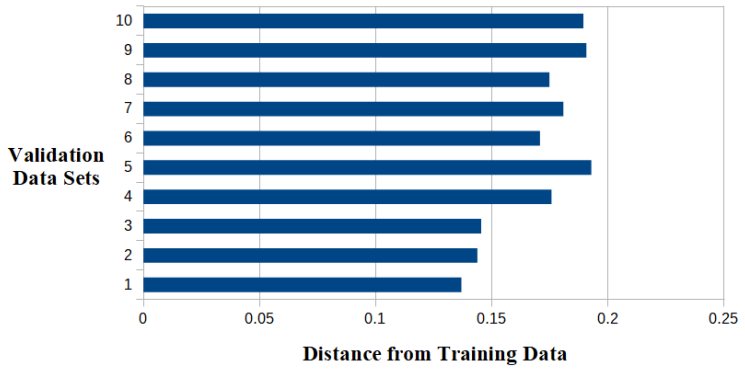



FIGURE 4.3. Differences for all Validation Sets from Training Data

#### 4.4.1.7 Cyber-Attack Distribution Results

Given the set of difference values between $f(T_0)$ and each of the validation data sets: $f(T_1)$, $f(T_2)$ ... $f(T_10)$, the mean and standard deviation values which were 0.1701 and 0.0208 respectively, were calculated. To detect the seeded password guessing attack, a threshold of two standard deviations above the mean is set, which was 0.2117. This defined threshold satisfies the empirical rule in the field of mathematics. Using Equation 4.2, it is illustrated that the distance for the attack data set from the normal data set is 0.2171 which is well above the defined threshold.

### 4.4.2 NoSQL Injection Attack

If an application were configured to use a database, like MongoDB, a typical NoSQL injection attempt would require a database query in which a WHERE clause is exploited. The hacker enters executable JavaScript that is triggered by the WHERE clause. For instance, if a user types the sequence $'jj'a' ==' a'$ in a username input field. If proper data sanitisation is not applied, this can result in the query returning data for all stored data objects in the database. In order for this to happen, all input format checks would have to be removed and ensure the query is implemented using the WHERE clause.

To detect this type of intrusion using distributed tracing, either the sequence of RPCs would need to be altered for the NoSQL Injection request or the duration of the RPCs would have to increase. This could be possible if the hacker could change a number of the microservice implementations. However, it is unlikely that the duration of the request would change even if a request for a single user or for all existing users was

sent off. In the case of all data objects, the result would be a returned set of objects called a ResultSet of batched size. The execution time for this returned set would not be appreciably longer than a request for a single data object.

Additionally, even if a ResultSet were returned, it would not be feasible to alter the source code to distribute the ResultSet throughout the application because of the extremely well-defined interfaces between the microservices in DeathStarBench. The microservices operation to return JSON user information is constrained to return only a single user object. It was concluded that in the case of a NoSQL Injection attack, even if it were possible to read multiple users from a database, the altered functionality would not be detected using distributed tracing.

## 4.5 Conclusion

The contribution of this chapter explored the application of distributed tracing to monitor and log API requests sent to a microservices application with the objective to support and enhance the detection of cyber-attacks. Anomaly detection is introduced and applied to detect seeded cyber-security attacks within the generated microservices application traces. Distributed tracing is typically used to carry out performance bottlenecks but to the best of our knowledge, this research project is the first in which distributed tracing has been applied to detect cyber-security attacks amongst microservices. It was concluded that it was possible to apply an anomaly detection method to detect an injected brute force password attack.

Because a password guessing attack is identifiable by examining the number of incorrect login requests, this technique is classified as group anomaly detection. To detect the seeded trace traffic, the base distribution for normal application traffic was calculated and compared to the distribution to that of a set of anomalous tracing data where the attack was seeded. By using a mean and standard deviation for the frequency distance from normal data to a series of validation data sets to set a threshold, it is notable that the distance between normal and anomalous data sets is greater than two standard deviations outside the mean. This result is a candidate for an anomaly detection technique as it satisfies the empirical rule.

It is also determined that this approach cannot be used to detect certain kinds of attacks targeting a microservices application. It is argued that sending a NoSQL Injection attack to return all user object information instead of a single user object would not cause any substantial alterations in the distributed traffic of the application and therefore would not be detectable.

A limitation of this approach is that using distributed tracing and frequency-based anomaly detection is that only the distributed relational behaviour of a microservices application using sequences of microservices calls is represented. The Euclidean distance metric defined in Equation 4.2 pays no attention to the order in which the sequential data occurs. To address this limitation, the plan is to represent microservice activity by using graph constructs where the nodes represent the microservices and edges of the graphs represent the calls to said microservices. Further research indicates that graph-related approaches have been previously used to model microservices and as well as to detect anomalous activity in such distributed applications [110] [111].

# Chapter 5

# Anomaly Detection by Traffic Forecasting using a DCRNN Model

## 5.1 Overview

In the previous chapter, cyber-attacks targeting an open source microservices application were investigated and anomaly detection approaches were proposed to detect cyber-security attacks amongst microservice application traces. It was specifically determined whether the irregular activity of cyber-attacks can be detected using anomaly detection based on the frequency distributions of microservice calls.

In this chapter the previous work to enhance the detection of cyber-attacks targeting microservices-based applications is extended. The third and final contribution is a deep learning methodology based on graph theory that is leveraged to detect the anomalous behaviour caused by cyber-attacks. It is proposed that due to their distributed system architecture, microservice application functionality can be modelled as a microservice call graph using graph theory. This graph representation can be augmented with microservice call traffic executed over a time series. Using this approach, the deep learning model can be trained to learn both the spatial and temporal dependencies of the microservice application traffic.

To model the microservice application tracing using the microservice call graph, the distributed behaviour of the microservices application is related to a diffusion process propagating across the call graph from a single node to its neighbours. The call graph will then be used to train a graph convolutional neural network (GCNN), called the Diffusion Convolutional Recurrent Neural Network (DCRNN), to discover the spatial and temporal dependencies of the microservice traffic over a time series of two minutes. The trained GCNN will then be used to predict future traffic activity using traffic forecasting. Threshold-based anomaly detection will then be applied to detect irregular RPC traffic within said call graph caused by the seeded cyber-attacks in microservices traffic.

For this contribution, the microservices application *SocialNetwork* from the open-source testbed, *DeathStarBench* is executed and active traffic is recorded using distributed tracing as in the previous chapter. Using the approach outlined above, we address the following research question.

*Can graph convolutional neural networks and traffic forecasting be used to detect cybersecurity attacks on microservices applications?*

## 5.2 Graph Theory

In the field of mathematics, graph theory can be defined as the study of graphs in all forms and sizes. A graph, on its own, collectively refers to a set of *nodes* connected by a set of *edges* [112]. An actual graph can be represented mathematically using the following equation.

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A}) \tag{5.1}$$

where $\mathcal{V}$ is a set of N nodes; $\mathcal{E}$ is a set of edges that are the pairwise connections between the nodes and $\mathcal{A} \in R^{NxN}$ is a two-dimensional array that represents the adjacency levels of each node to each other. Let $v_i \in \mathbf{V}$ denote a node at index $i$ in a where $e_{i,j} = (v_i, v_j) \in E$ represents an edge from $v_i$ to $v_j$.

The application of graph theory can be used to model a dynamic system of actual entities where the nodes represent the entities and the edges represent the connections

between said entities. The resulting graph model is then suitable for monitoring the pair-wise relationships between entities and providing an intuitive interpretation of these dependencies. Directions can be applied to the edges to produce a **directed graph**. A directed graph can be used to represent a road network (where one-way lanes can be modelled), hyperlinks between web pages while web-browsing and existent food chains between prey and predator species.

## 5.3 Traffic Forecasting

Traffic forecasting is defined as being able to predict future activity given previously learned traffic data derived from a network-based model [42]. The motivation behind traffic forecasting is the question *"Where are we and where are we going?"*. The concept has been applied heavily to transportation systems for drivers on the road as timely traffic prediction can be crucial to anticipate incoming traffic conditions for traffic control and guidance [113] [37].

## 5.4 Methodology

In this section, the implementation of the DCRNN model [42] and the group-based anomaly detection to discover cyber-attacks in a microservices application used for this work are presented.

### 5.4.1 Microservice Traffic Generation

In the first step of this methodology, the same microservice application from the DeathStarBench suite, *SocialNetwork* is executed as in Chapter 4, and generated synthetic data sets comprised of requests from the application functionality through penetration testing. These data sets are available at [114]. In microservices, RPCs are produced between different microservices in which one microservice initiates RPCs to one or more others, providing collaborative functionality.

### 5.4.2 Directed Graph Representation

In this approach, the topology of a microservices-based application is modelled as a directed graph. The dependencies between RPCs in an application can be represented as source-destination pairs which represent nodes on the previously mentioned call graph. Whenever nodes share a source or destination, the corresponding edge is assigned a weighted value.

Using this approach, a weighted directed graph **G** is returned which can be represented mathematically using the Equation 5.1 defined in Section 5.2 which returns the following as listed:

- **N**: a set of all unique RPC source-destinations pair nodes
- **E**: a set of all edges formed between the RPCs when said nodes share a source or destination
- $\mathbf{A} \in R^{NxN}$: a weighted adjacency matrix that represents the level of adjacency for each node to each other

When an RPC connection exists between two source-destination pair nodes, **a** and **b**, they are assigned a weighted value accordingly. Each relation is assigned the weight values as follows: when **a** and **b** share the same source or destination value, a weighted edge from both **a** to **b** and from **b** to **a** exists with an assigned weight value of 0.5. In the case of one node's source value being another node's destination, there is a dependency edge with a weight value of 1.0. This approach for assigning weighted values to RPC connections is illustrated in Figure 5.1.

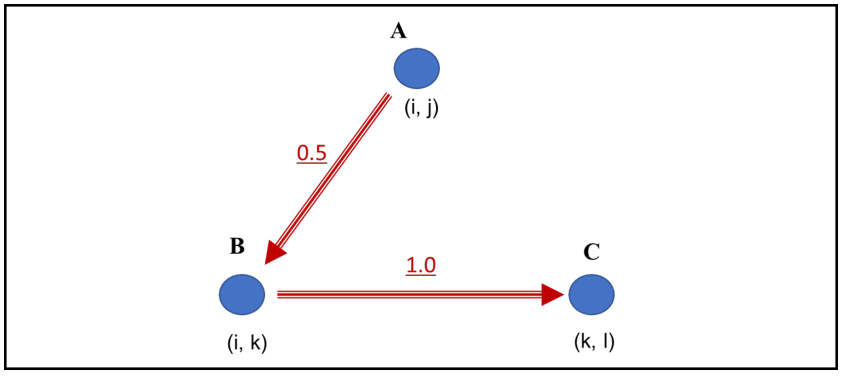



FIGURE 5.1. Time Series Traffic

This approach promotes scalability, reusability and supports the distributed architecture and the different dependencies between the microservices. The computing procedure for building this adjacency matrix $\mathcal{A}$ is defined in Algorithm 1:

**Algorithm 1:** Construct the Adjacency Matrix.

**Input:** An RPC Node Set: $N$
**Output:** The adjacency matrix: $\mathcal{A}$

```
A ← empty matrix(shape = len(N) * len(N));
for i ←0 to len(N) do
    for j ←0 to len(N) do
        if N[i].src == N[j].src or
        N[i].dst == N[j].dst then
            A[j, i] ← 0.5;
            A[i, j] ← 0.5;
        end
        if N[i].src == N[j].dst then
            A[j, i] ← 1;
        end
        if N[i].dst == N[j].src then
            A[i, j] ← 1;
        end
    end
end
```

### 5.4.3 Representation of Microservices Traffic Matrix

When a request call to a microservice is initiated and logged, additional attributes are also monitored. Given the existing directed graph **G** defined in Section 5.4.2 and a set of all existing nodes $N$, a set of attributes $M$ is returned for every existing node. These attributes per node can be represented as a $NxM$ dimension matrix. For this project, only a single attribute is represented concerning the stored application traffic. Simply put, this value is the number of times an RPC pair-value is invoked within a particular time window. The definition for how this application traffic matrix is obtained is as follows.

As the application executes along a time series of two minutes, a set of $T'$ specified time steps is defined. These time windows will be of equal duration for the simplicity of the experiment. This methodology iterates throughout the entire data set of logged RPC calls and computes the traffic for each node within each specified time window. This computed data can be compiled into a 2-dimensional array stored as a data file. Given

this RPC traffic file and the directed graph **G**, a series of application traffic matrices at every time step is returned. This sequence of time series ranges from $X_{t-T'+1}$ to $X_t$, where $X_t$ denotes the traffic matrix **X** at time step **t**.

### 5.4.4 Traffic Forecasting Formula for Microservices

The aim of the traffic forecasting problem in regards to the microservices application is to define a function to predict microservice traffic at future time steps. Given the directed graph **G** defined in Section 5.4.2 and the established time series of *T'* traffic matrices from Section 5.4.3, the resulting function is outlined in Equation 5.2:

$$[X^{(t-T'+1)}, ..., X^{(t)}; G] \xrightarrow{h(.)} [X^{(t+1)}, ..., X^{(t+T)}] \tag{5.2}$$

where *T'* RPC graph signals at time step **t** are mapped to *T* time steps in the future. This time series of RPC signals are shown below in Figure 5.2.

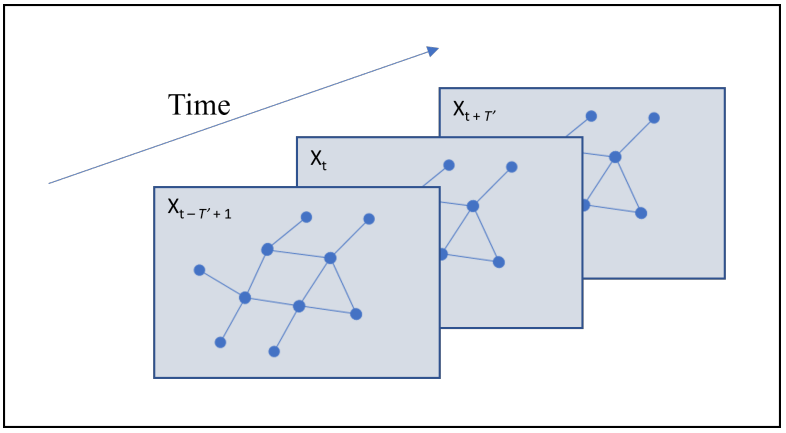



FIGURE 5.2. RPC Graph Traffic over a Time Series

### 5.4.5 Spatial Dependency Modelling

Graphs are immensely complex structures, particularly directed graphs because of the differing layers of neighbouring nodes and levels of adjacency. The meanings for these complex constructs can be learned using a class of CNN models called GCNNs.

For this approach, the DCRNN model is trained to learn the RPC traffic which can be modelled on the directed graph **G** from Section 5.4.2. One approach for said network to

learn the data representations of $\mathbf{G}$ is to model the spatial dependencies and thus learn the stochastic features of the traffic, similar to the work of [115]. This can be done by relating the existing RPC traffic as a diffusion process [33]. A example of this approach can be likened to a simple walkabout from one node to its neighbour on $\mathbf{G}$. This process can be expanded where active RPC traffic flows from a single node to its neighbouring nodes can be represented as a weighted distribution of infinite random walkabouts that propagate throughout graph $\mathbf{G}$. The stationary distributed form of said diffusion process [116] can be represented in closed form as defined in Equation 5.3:

$$\mathcal{P} = \sum_{k=0}^{\infty} \alpha(1-\alpha)^k (D_O^{-1} W)^k \tag{5.3}$$

where $k$ represents the diffusion step. Our aim is to define a *diffusion convolution* operation where active traffic flow can be modeled over a traffic matrix signal $\mathbf{X} \in \mathbb{R}^{NxP}$ where $P$ represents the 1-dimensional traffic input value. This operation will also support a diffusion process in the reverse direction from a destination node back to its source so the model can learn dependencies from both upstream and downstream traffic. Given traffic attribute matrix signal $\mathbf{X}$ and a kernel of filter $f_\theta$ represented as $\star\mathbb{G}$ [43], the diffusion convolution operation is defined in Equation 5.4 as follows:

$$\mathbf{W}_{\star G}\mathbf{X} = \sum_{d=0}^{K-1} (W_O (D_O^{-1} A)^d + W_I (D_I^{-1} A)^d) X \tag{5.4}$$

where $K$ is the specified maximum number of diffusion steps allowed; $W_O$ and $W_I$ represent the learnable filters for the bidirectional diffusion process; $D_O^{-1}$ and $D_I^{-1}$ are the state transition matrices for the upstream and downstream diffusion processes respectively, $\mathcal{A}$ is the weighted adjacency matrix defined in Section 5.4.2, and $D_I$ and $D_O$ represent the in-degree and out-degree matrices that provide the capability to learn from the bidirectional processes modelled on directed graph $\mathbf{G}$. Using this convolution operation, a **Diffusion Convolutional Layer** is built which is trained to map $P$-dimensional inputs to $Q$-dimensional outputs denoted as $\mathbf{H} \in \mathbb{R}^{NxQ}$ as outlined by Equation 5.5.

$$\mathbf{H}_{:,q} = \alpha(\sum_{p=1}^{P} X_{:,p} \star \mathbb{G} f_{\Theta_{q,p,:,:}}) \quad \forall \quad q \in \{1, \ldots, \mathrm{Q}\} \tag{5.5}$$

where $\alpha$ refers to an activation function for mapping the input matrices to the outputs (e.g. ReLu, Sigmoid); $\Theta \in \mathbb{R}^{QxPxKx2}$ represents the tensor representation for the trainable parameters and $f_{\Theta_{q,p,:,:}}$ are the actual filters. This **Diffusion Convolutional layer** described above is trained to capture the actual graph-structured data representations using the stochastic gradient based methodology [117].

### 5.4.6 Temporal Dependency Modelling

The DCRNN model is leveraged to capture the temporal dependencies in the microservice traffic by implementing an RNN's capability to learn from previous observations in sequences. The actual model is comprised of an encoder-decoder framework where both components are RNNs. It was proposed to use a simpler and more-widely-used RNN model called the Gated Recurrent Unit (GRU) [118] as the primary building block of the proposed DCRNN, following the work of [42]. To support traffic forecasting defined in Equation 5.2, a sequence-to-sequence architecture [119] was employed for the encoder-decoder framework. Given a time series of traffic matrices $\mathbf{X}$, the encoder component takes as input the time series, the data is translated into a vector representation and its final states are used to initialize the decoder. Subsequently, the decoder component reads from the vector representation and generates data predictions given previously learned ground truth observations.

During testing, ground-truth observations will be replaced by generated output predictions which could cause degraded model performance. To mitigate these discrepancies, scheduled sampling [120] was employed to set a probability-based threshold to determine whether a value is a ground-truth input or a model predicted output at a previous time step. This value will decrease continuously during training. This scheduled sampling will use an inverse sigmoid function for its probability value $\mathbf{p}$ as defined in Equation 5.6:

$$\mathbf{p} = \frac{\mathbf{c}}{\mathbf{c} + \exp(\mathbf{i}/\mathbf{c})} \tag{5.6}$$

where $\mathbf{c}$ is the number of steps for controlling the decreasing ratio and $\mathbf{i}$ represents the number of iterations the DCRNN is trained for.

To build the DCRNN model, a GRU unit was taken and the matrix multiplication functionality was replaced with the diffusion convolution operation defined in Equation 5.4. This new design leads to the *Diffusion Convolution Gated Recurrent Unit* (DCGRU) [42]. In Equation 5.7, the functionality that denotes the DCGRU cell is defined as follows:

$$\begin{aligned} \mathbf{r}^t &= \sigma(\Theta_{r\star\mathbb{G}}[X_t, H_{t-1}] + b_r) \\ \mathbf{u}^t &= \sigma(\Theta_{u\star\mathbb{G}}[X_t, H_{t-1}] + b_u) \\ \mathbf{C}^t &= \tanh(\Theta_{C\star\mathbb{G}}[X_t(r_t \odot H_{t-1}] + b_C) \\ \mathbf{H}_t &= \mathbf{u}^t \odot H_{t-1} + (1 - \mathbf{u}^t) \odot c_t \end{aligned} \tag{5.7}$$

where $X_t$ and $H_t$ denote the input and output traffic matrix at time step $t$ respectively; $\mathbf{r}^t$, $\mathbf{u}^t$ and $\mathbf{C}^t$ denote the reset, update and cell state at $t$; $\odot$ represents the element-wise tensor multiplication; $\star\mathbb{G}$ refers to the actual diffusion convolution operation defined in Equation 5.4; $\Theta_r$, $\Theta_u$ and $\Theta_C$ represent the corresponding filters applied to each diffusion convolution equation, and $b_r$, $b_u$ and $b_C$ denote each filter's respective biases.

Given the encoder-decoder architecture, both the RNN components are composed of Diffusion Convolution layers defined in Equation 5.5 built with DCGRU, the model is trained using Backproagation through time, and the sequence-to-sequence architecture and scheduled sampling are also integrated. These layers allow the model to be trained with sequential data over the historical time series and learn the long-term temporal dependencies of the data.

The basic architecture of the DCRNN, including the encoder-decoder framework is illustrated in Figure 5.3

### 5.4.7 DCRNN Architecture & Training

Given the spatial and temporal dependency modelling techniques outlined in Sections 5.4.5 and 5.4.6 respectively, the completed DCRNN model is trained to learn both the spatial dynamics of the weighted adjacency matrix $\mathcal{A}$ defined in Section 5.4.2 and the temporal dependencies of the aforementioned time series from 5.4.6 at the same time.

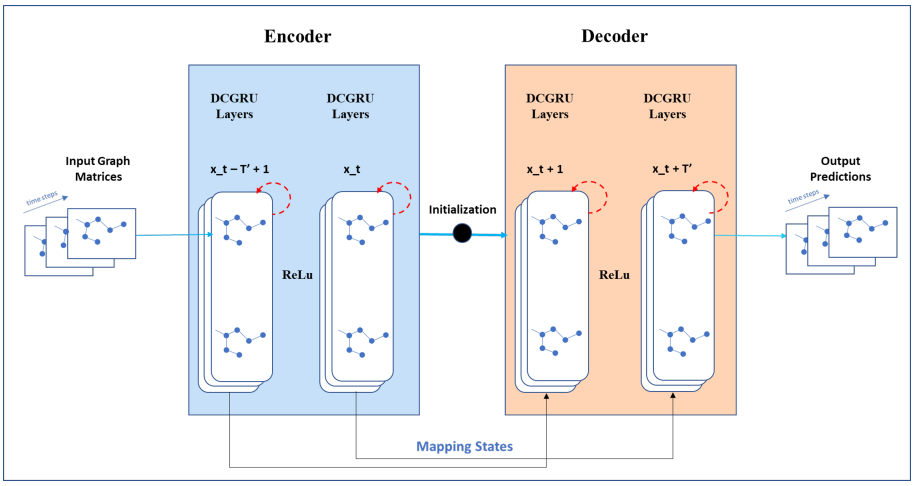



FIGURE 5.3. System Design of the Diffusion Convolutional Recurrent Neural Network

Both the encoder and decoder are RNNs composed of Diffusion Convolutional layers built with DCGRU units. The encoder takes as input the historical time series of traffic matrices and its final states are used to initialize the decoder. The recurrent layers are trained using backpropagation through time over a number of iterations and the RNNs use a *ReLu* activation function to map input matrices to output matrices. The decoder then outputs predictions based on ground-truth values.

During the training phase, the time series of the microservices traffic matrices and the adjacency matrix $\mathcal{A}$ are fed into the DCRNN model as input. The model trains to discover and learn the temporal dependencies along the time series using the encoder-decoder architecture of the model which are RNNs that train using backpropagation though time over a number of iterations. The input data is fed into the encoder and is trained by the model over a number of epochs. The final state of the input data is used to initialize the decoder component. The decoder then generates output matrix predictions over $T$ future time steps.

$$\textbf{MAE} = \frac{1}{s}\sum_{i=1}^{s}|y_i - \hat{y}_i| \tag{5.8}$$

### 5.4.8 Anomalous Microservices Detection

Given the microservice RPC traffic predictions returned by the DCRNN model, anomaly detection is performed to discover irregular RPC traffic at a specified time window. This application of anomaly detection is loosely similar to the application used by [11]. This is applied by setting a threshold value on the prediction error of the testing RPC traffic. The prediction error $E$ is the absolute difference between the ground-truth values and the model's predicted values as denoted by Equation 5.9:

$$E_i^t = X_i^t - \tilde{X}_i^t \tag{5.9}$$

where $X_i^t$ represents the RPC node value $i$ at time step $\mathbf{t}$ and $\tilde{X}_i^t$ represents the output prediction for the same values. For every node $i$, $H_i$ represents both the upper and lower limits for their respective distribution of thresholds and are calculated as follows:

1. calculate the mean $\mu_i = \frac{1}{n}\sum_{t=i}^{n} x_i$
2. calculate the standard deviation $\sigma_i = \sqrt{\frac{(x_i-\mu_i)^2}{n}}$ for $E_t$
3. set the thresholds using the formula $H_i = \mu_i \pm (2 * \sigma_i)$

## 5.5 Experiment

In this section, three experiments with the microservices application *SocialNetwork* from the DeathStarBench suite are described, the preparation of the application for the experiment was outlined and three types of cyber-attack are described which are simulated against the application in each experiment through penetration testing. The microservices application was run using the same software environment and tools as the experiment in Section 4.3 in chapter 4. For each experiment, the microservices traffic data was logged using distributed tracing, the data was prepared and the DCRNN was trained to learn the prepared data and evaluated as outlined in Section 5.4.7 and the threshold-based anomaly detection from Section 5.4.8 was applied to detect the injected cyber-attacks within the microservice traffic. The objective is that this approach can detect these attacks as a group anomaly due to the large irregular quantity of distributed traces used to comprise these attacks in a short time window. This research proposal and the end results and determinations are documented in the publication [121].

### 5.5.1 Microservices Data Preparation

Once the microservices application, *SocialNetwork* was running, initial work was performed to construct and operationalize the topology of the social networking application. To start the experiment, 962 virtual users were registered with *SocialNetwork* and 18800

*user-follow* relationships were established amongst them. Finally, a directed graph was defined where the nodes represent the users and the edges represent the following relationships. Subsequently, API request calls to *SocialNetwork* were sent using HTTP workload traffic generators and hard-coded scripts written in the Python 3.7 programming language. The implementation of *SocialNetwork* used for this work is available here [122].

In the course of the experiment, *SocialNetwork* was run once during three separate experiments each with a different seeded cyber-security attack. For each experiment, both normal application traffic and irregular traffic caused by the injected cyber-attacks were generated. In the microservices traffic data, normal activity was generally composed of calls with the API request *wrk2-api/post/compose* to compose and upload a text to the social network application. The RPC nodes for the **composePost** functionality are listed in Table A.4. Other microservices activity in the experiment include API calls *wrk-api/user/register* to register new users; *api/user/login* for a user to log into their account and *wrk2-api/user-timeline/read* to look up a user's application activity.

For each experiment, synthetic data sets of distributed traces caused by microservice activity sampled over two minutes were generated. This resulted in approximately 18500 microservice calls and a vocabulary of 63 unique microservice source-destination pair nodes which are listed in Appendix A. Using this vocabulary of RPC pair nodes, a representation of the application's topology in the form of a directed graph $\mathbf{G}$ outlined in Section 5.4.2 was constructed and developed a weighted adjacency matrix $\mathcal{A}$ was developed to represent each RPC pair node's levels of adjacency to each other as defined in Algorithm 1. A series of 100 time windows was extracted from the synthetic data set of microservice traces. Using this set of time windows, the traffic for every RPC node was computed for every time window. For the training stage of the experiment, 85% of the computed traffic was set aside while the remainder was used for the testing stage. To detect the cyber-security attacks, the RPC traffic in the time window where the cyber-attack was injected was analysed.

### 5.5.2 Specifications for DCRNN Model

For these experiments, the DCRNN model utilized was composed of two Diffusion Convolution Recurrent Layers defined by Equation 5.5 built with DCGRU as defined in

Section 5.4.6. Each layer contains 150 units and is configured with the bidirectional diffusion convolution operation from Equation 5.4. Using the traffic forecasting approach specified in Equation 5.2, the DCRNN model predicts the future RPC traffic matrix at a single following time step. Using a dual-random walk filter type to model the temporal parameters, the bidirectional diffusion process in Equation 5.3. The adjacency matrix $\mathcal{A}$ defined in Section 5.4.2 is also fed into the DCRNN as a trainable parameter to load the spatial graph data and learn the levels of adjacency between the RPC pair nodes.

During the training process, the hyper-parameters for the DCRNN model are as follows: the initial learning rate = 0.01; the decay ratio for the learning rate = 0.1; the maximum diffusion step = 2; the *Adam* optimizer was used and 1000 decay steps for the scheduled sampling formula. The actual model was trained over 50 epochs with a batch size of 16 and early stopping set after 15 epochs.

The performance of the DCRNN model was evaluated by computing its MAE metric defined in Equation 5.8 as a measure of the model's prediction error. The aim was to investigate the effect of spatial and temporal modelling. To carry out this evaluation, 70% of the data set was set aside to calculate the training loss and the remainder for the validation loss. The learning curves for the training and validation MAE metrics are illustrated in Figure 5.4.

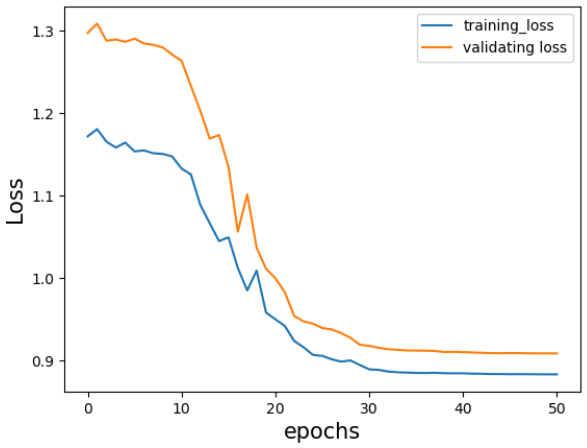



FIGURE 5.4. MAE loss values for Training and Validation Data Sets

As observed in Figure 5.4, the loss values start out as moderately high, before gradually lowering until both metrics stabilize at 30 epochs and flatten out for the remainder of the training duration. It is worth noting that while both metrics converge to a point of

stability, the loss gap between the loss values is minimal with a divergence lesser than 0.1 in measurement. This shows that the DCRNN has proven to be reasonably suitable for the data set.

### 5.5.3 Simulated Cyber Attacks against SocialNetwork Application

To seed various cyber-attacks amongst the synthetic data sets, penetration testing is used to simulate a different cyber-attack for each experiment and analyse the effect each attack has on the application. To inject each attack into the microservices RPC traffic data, hard-coded scripts were executed to send user API requests that would resemble the specific cyber-attack as they would appear in real-life.

#### 5.5.3.1 Brute Force Password Guessing attack

The *SocialNetwork* application sports a API request call for a user to log into their account, *api/user/login*, so it is feasible to simulate a brute force password guessing attack against the application. In the application architecture, the API call is delegated to the microservice *user-service*. The microservice API calls that execute in response to a *api/user/login* call are previously described in Section 4.4.1.2 of chapter 4 and the RPC nodes related to this login functionality are listed in Table A.1 found in Appendix A.

A brute force attack is used to hack a user's account using any possible combination of letters, numbers or any other keyboard characters through trial-and-error. This results in hundreds of login requests per second [123]. In this experiment, a simulated brute force attack is injected into the testing data set of microservice traces at a time window **t**. This seeded attack is performed by entering every possible keyboard character as a single character password to crack an existing account. The resulting microservice traffic at time window **t** is populated with ninety-two incorrect login API calls. The anomaly detection formulae from Section 5.4.8 was then applied to detect the irregular frequency of login microservice RPC nodes. To distinguish the irregular login activity caused by the brute force attack from the normal application functionality at time step **t**, the prediction error $E_i$ for a set of select eight RPC nodes and their respective thresholds $h_i$ were calculated. Four of these nodes represent the login functionality and the remaining four nodes represent the normal traffic to compose and upload users' posts.

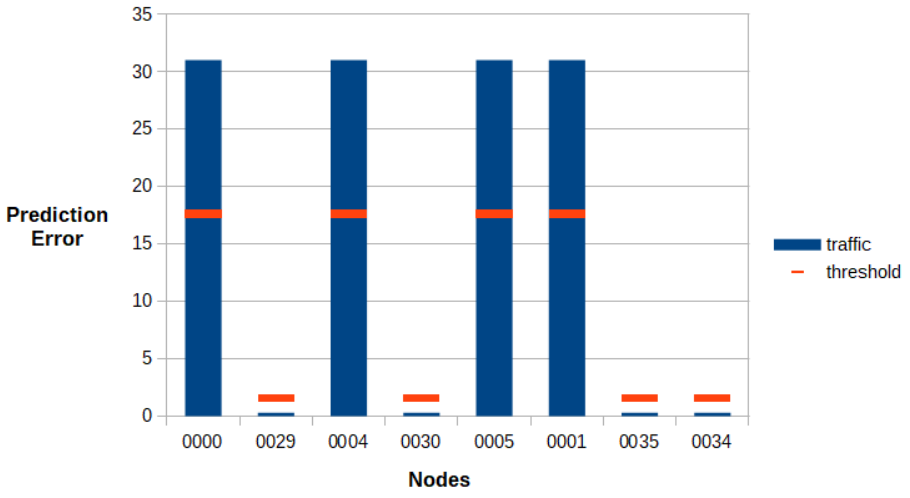



FIGURE 5.5. Brute Force Password Guess Attack

From Figure 5.5, it is inferred that $E_i$ values for a set of select RPC pair nodes at time step **t** exceed their respective $h_i$. These nodes which exceed their threshold represent the irregular login functionality executed during the brute force attack, and are described as follows:

- **0000**: the API call *api/user/login* is sent by a client user to the application
- **0001**: the API request is directed to the *Nginx* load balancer
- **0004**: the application recognizes a user object whom has previously logged in and calls the operation **Login**
- **0005**: application checks if credentials are cached and calls **MmcGetLogin**

As shown in Figure 5.5, the prediction error of the RPC traffic for each of the four **userLogin** nodes exceed their defined thresholds. In comparison, the four random nodes from the **composePost** functionality do not meet their set thresholds. Therefore, it was determined that the brute force attack seeded at time step **t** was detected using the anomaly detection means.

#### 5.5.3.2 Batch Registration of Bot Accounts

In *SocialNetwork*, the API request *wrk- api/user/register* is called to create a new user profile. Therefore, it is feasible to send a multitude of these requests to the application within the time span of a second to seed the creation of a set of bot accounts in the microservices traffic. Like the **userLogin** functionality, the application delegates the request to the microservice *user-service* and the operations **RegisterUserWithId** and **MongoInsertUser**. The RPC nodes related to this functionality are defined in Table A.2.

During a batch registration, bot accounts are typically created in tens or hundreds [13]. In this experiment, multiple API requests are sent to the application to create 100 new accounts to emulate an attempted batch registration and seeded the virtual attack at time step **t** likewise with the brute force attack. By calculating the prediction error $E_i$ and the calculated threshold $h_i$ for eight randomly selected RPC pair nodes at time step **t** were set aside for analysis. $E_i$ for these nodes were composed of the account registration traffic caused by the batch registration attack and normal traffic caused by the regular application functionality. These prediction error values and their respective defined thresholds are displayed in Figure 5.6.

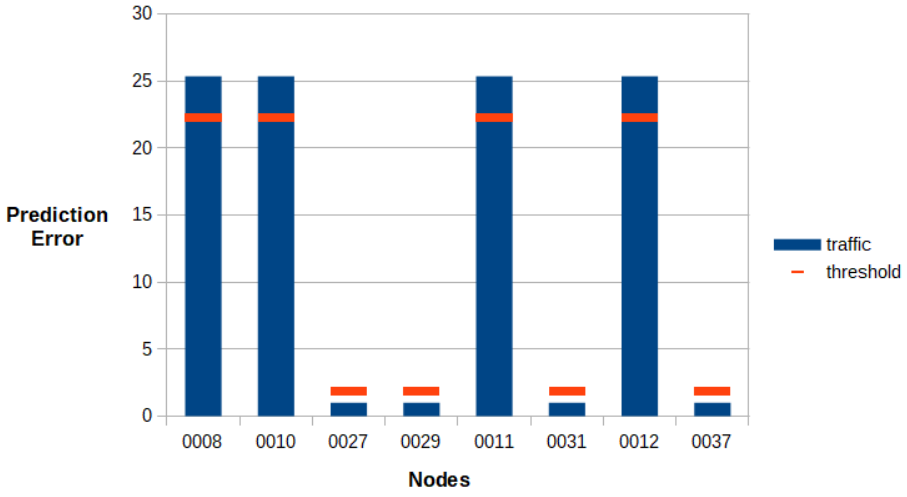



FIGURE 5.6. Batch Registration of Bot Accounts

From Figure 5.6, it is inferred that $E_i$ for the RPC nodes which denote the irregular **userRegistration** functionality during time step **t** exceed their respective $h_i$. These nodes are outlined as follows:

- **0008**: the API call *wrk2-api/user/register* is sent to the application
- **0010**: the Nginx load balancer calls its local **RegisterUser**
- **0011**: the API call is delegated to the microservice *user-service* and the operation **RegisterUserWithId** is called
- **0012**: the operation **MongoInsertUser** is called to register a new user in MongoDB database

It was determined that the irregular $E_i$ for the random nodes representing the account creation functionality were the result of the registration of bot accounts, determining that the batch registration was successfully detected using the anomaly detection formulae.

#### 5.5.3.3 Distributed Denial-of-Service attack

The third cyber-attack investigated for this work is an application-layer Distributed Denial of Service (DDoS) attack. The *SocialNetwork* application supports an available GET request using the API request *wrk2-api/user-timeline/read* which returns the timeline of a user's application activity. A user's stored timeline would include the composition and uploading of text, images and other media to the application. This API request is forwarded to the microservice *user-timeline-service* and the responding microservice operations called include **ReadUserTimeline** and **MongoFindUserTimeline**. The RPC nodes defined for these operations are defined in Table A.3.

Application layer DoS attacks have a magnitude of between 50 and 100 requests per second [124]. For this experiment, a HTTP Flooding attack is simulated by seeding 100 HTTP GET requests to return a user's application timeline. This would simulate a Denial-of-Service intrusion targeting *SocialNetwork*. In a real-world scenario, the HTTP Flooding attack would cause a disruption to the microservice *user-timeline-service* and subsequently hinder the following operations **ReaduserTimeline**, **RedisFind** and **MongoFindUserTimeline** thus disrupting the storage and caching of said user timelines. After the prediction error values for all RPC nodes at time step **t** were calculated, eight nodes were selected for analysis. Four of said nodes represent the prediction error traffic caused by the DDoS attack while the traffic values for remaining

nodes represent the regular functionality of the application. The prediction error values and the defined thresholds for these nodes are displayed in Figure 5.7.

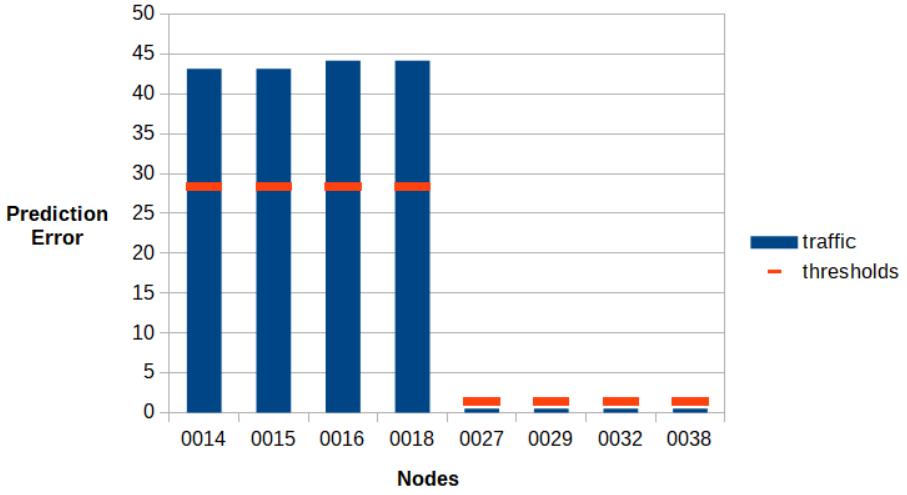



FIGURE 5.7. HTTP GET Flooding Attack

As observed in Figure 5.7, the prediction error traffic for a select number of nodes related to the functionality for the GET request at time step **t** is shown to exceed their defined thresholds. These individual nodes are outlined as follows:

- **0014**: the client user sends the *wrk2-api/user-timeline/read* API call to the application's web server provided by *Nginx* and subsequently calls its child node denoted as **0015**
- **0015**: the web server calls the operation **ReadUserTimeline** and delegates the API call to the microservice *user-timeline-service*
- **0016**: the API request calls the operation **ReadUserTimeline** from the actual *user-timeline-service*
- **0018**: the microservice calls an operation **MongoFindUserTimeline** to return a user's timeline activity stored in MongoDb

Based on the observations shown in Figure 5.7, it was determined that the anomalous RPC traffic to return a user's application timeline caused by the simulated DDoS attack was detected.

## 5.6 Conclusion

The contribution for this chapter is a continuation of the objective from the previous chapter to explore the application of anomaly detection methods to microservices application tracing to support the detection of cyber-attacks. Like in the previous contribution, distributed tracing and anomaly detection were implemented to discover cyber-attacks targeting a microservices application. The aim of this contribution was to establish that the distributed behaviour and topology of a microservices-based application could be modelled using a graph theory representation, a microservice call graph, that a deep learning model could be trained to learn the relational dependencies and attributes of these representations and subsequently make predictions of future microservice activity using traffic forecasting.

This methodology was outlined as follows: a state-of-the-art deep learning model the Diffusion Convolutional Recurrent Neural Network (DCRNN) was leveraged to discover and memorize the spatial and temporal dependencies of the microservice traffic. Traffic forecasting was then used to predict future microservice activity at a future time step along a time series. The performance of the DCRNN model was evaluated by applying a threshold-based anomaly detection methodology to detect irregular microservice activity that indicates seeded cyber-attacks against the microservices application.

Using this outlined approach, the *SocialNetwork* microservice application was executed, it was determined that three particular types of cyber-attacks can be detected against the application: including a brute force attack, a registration of bot accounts and a HTTP Flooding attack. For each attack, the anomalous microservice activity was compared to the regular microservices activity at run-time. To apply the threshold-based anomaly detection in terms of RPC traffic volume, the mean and standard deviation were calculated. Because of the irregular volumes of unlabelled RPC traffic to simulate these virtual attacks, the anomaly detection method can be categorized as an unsupervised method and a group anomaly detector. It was also determined that the resulting RPC traffic at the time slice caused by each seeded cyber-attack was proven to be greater than two standard deviations outside the mean which is satisfactory with regard to the empirical rule. Therefore, it was determined that this approach through the application of traffic forecasting supports the detection of cyber-attacks.

### 5.6.1 Limitations

It should be stressed that this work is loosely similar to that of [11] which trains multiple DCRNNs to learn a different subsystem of the application's functionality. However, a limitation of using multiple models is that their methodology cannot detect cyber-attacks that could span across multiple application subsystems. The approach here is novel in that only a single DCRNN is trained to learn the entire application. This approach is generally necessary as it can be difficult to determine where an attacker will target the application and what functionality they intend to compromise. A potential drawback to this approach however is that it can be rather difficult to maintain such a large and unified model for the entire application. Microservices are updated and old functionality is deprecated regularly, so there is a need to re-train the entire model. This costs time and resources.

Though this distributed tracing-based approach is effective in detecting volumetric cyber-attacks targeting the application, it works by monitoring the traffic flow from the HTTP API calls which are sent over the application-layer of the system. During an attack called Man-In-The-Middle (MITM), WiFi hotspots are constructed over the network layer to intercept confidential information from transactions between users and applications. This form of cyber-attack would occur at the network layer of the application. Therefore, this form of cyber-attack would not be detected by this methodology. Another type of cyber-attack that the high-volume traffic detector would not be able to detect is a cross-site scripting attack. In the *SocialNetwork* application, a user could compose a post showing a web link to a malicious site that an honest user clicks on forwarding the user to the site resulting in the attacker stealing the user's privileged credentials. Because this single post only requires a single HTTP API call to be sent to the application, this attack would not be detected by this group anomaly detector approach.

In this proposal, the anomaly detector is data-driven and relies heavily on machine learning. The machine learning model could be compromised if the model itself and the actual data are not protected. The threshold value for anomaly detection is derived by minimizing the prediction error value for every RPC node at run-time. A potential adversarial machine learning attack which violates the integrity of the model at testing time can be carried out when the DCRNN model is operational and the anomaly

detection mechanism has been deployed. During this attack, the attacker attempts to subvert the anomaly detector by modifying ground truth values so to not meet the set threshold. These malicious ground-truth values would not meet the criteria and irregular traffic data would not be detected as anomaly. A possible countermeasure against this the adversarial machine learning attack is to smooth the model's decision boundaries and the effects of the attack would be reduced [125].

Another form of adversarial machine learning attack that can be used against the machine learning model is poisoning [126] the actual model to perturbates training data at run-time that consequently alters the threshold setting for the anomaly detector. The appropriate defences can be met by protecting both the machine learning model and the data. A countermeasure against this attack is to train the machine learning model using data from randomized sources which leaves it difficult for the attacker to devise an effective adversarial attack [127]. Data sets and machine learning model can also be protected using sanitization. For data, a sanitization process detects and removes anomalous samples based on predefined conditions [128]. In the case of the model being implemented for a software enterprise system, the model is continuously trained using newer data separately from the system, therefore mitigating the impact of poisoned samples. In the case of the microservices application, santitization may not be possible due to a lack of available test beds and defining such rules can be difficult.

# Chapter 6

# Conclusion & Future Works

## 6.1 Conclusion

Due to the ubiquity of continuous software development, all aspects of software computing are growing increasingly sophisticated especially the worldly issue of cyber-security. Because of the constant development of immensely complex cyber-threat models, software organizations are being targeted every day. As a result, more sophisticated countermeasures against such cyber-threats are in high demand.

One proposal for this worldly conundrum was for cyber-security personnel to employ a defined model that represents their software enterprises' general application process. This model would be used to provide an improved cyber situational awareness of their application process model. Advantages of this model include being able to detect and pre-empt impending cyber threats to their application. This allows the cyber-defenders to enable defensive measures and allocating appropriate resources to address the threat. For this research project, we explored how application process tracing can be applied to support and enhance the prevention and detection of cyber-attacks. Two research objectives proposed for this project were 1) to investigate how process flow prediction in application process threads can improve cyber situational awareness and 2) to explore how anomaly detection approaches can be applied to microservice process traces to detect cyber-attacks.

For the first contribution of this project, we proposed the following research question:

1. *Can process mining and deep learning models predict next steps in an application process?*

Initially, process mining algorithms using ProM, a process mining tool, were explored to discover the overall framework of application processes. Process mining plug-ins were executed each returning a defined process model. These output graphic representations related to an organisation's mission or infrastructure which provides a contextual oversight of existing business processes. This supports the capability to observe and identify future events in this process model. The process mining algorithms, the $\alpha$ algorithm and the IM were applied to sample data sets to illustrate process-oriented work-flows. The IM, in particular, was used to model the existing outline of an existing cyber-attack. However, it was established that the process mining tools lacked a quantifiable means to perform process flow prediction.

Therefore, for addressing our first research question, we directed our attention to training a deep learning model to learn the sequential form of process instances or cases to discover a contextual oversight. This deep learning model evaluated the ability to predict future events in ongoing cases. In a case of differing events, events at certain time steps of a case affect the remainder of the case. To learn these long-term dependencies within the sequential data, we built, trained and evaluated an LSTM RNN network model. Two different methodologies of building this DL model were used: a *Prefix*-based approach and a *Teacher forcing* approach. The two different models were evaluated using four different data sets and their ability to predict future events was calculated in terms of accuracy. The different experiments with our LSTM model highlighted that several factors of the data set affect the model's performance in predicting future events.

1. the size of the data set
2. the number of event types in the vocabulary
3. the frequency distribution of the event types
4. the size, shape and architecture of the model

Therefore, by being able to perform process flow prediction using our defined LSTM model, we were able to identify future process events, thus discovering a contextual

oversight of the overall general application process and therefore, cyber-security personnel could gain an improved cyber situational awareness of the process by being able to pre-empt cyber-threats which supports cyber-attack prevention.

The subsequent research question is:

2. *Can frequency distribution-based anomaly detection detect cyber attacks in a microservices application?*

For this contribution, we directed our attention towards applying anomaly detection techniques to microservice application traces to enhance the detection of cyber-attacks. In microservices, calls to these services are executed in response to an end user's HTTP API call. To examine microservice activity, an open-source microservices application known as *SocialNetwork* from an available benchmark suite named *DeathStarBench* was run and calls sent to this application were logged using a distributed tracing tool called Jaeger. Our hypothesis was that the presence of a cyber-attack generates high microservice traffic volumes which could be discovered using a group anomaly detection approach, thus detecting the cyber attack.

First, distributed tracing and threshold-based anomaly detection were used to detect irregular distributions of traces. Secondly, we generated normal application traffic using distributed tracing and calculated the frequency distribution of unique microservice traces which represent different functionality of the application and derive a base distribution from this regular data. Subsequently, we aim to detect cyber-attacks in the application traffic by calculating the frequency of microservice traffic resulting from the attack and identifying it as an anomaly if the traffic sufficiently differs from the base distribution of normal data. This difference between the regular distribution data and the testing data set was captured using a Euclidean distance metric.

During data generation, microservice calls that belong to the same distributed trace are identified by a unique identifier assigned by Jaeger. Through a UI provided by Jaeger, users could also observe these traces and microservice calls. This AI-based approach was evaluated by seeding a password guessing attack and monitoring the irregular application logic for logging into an account. This approach was proven to be successful as the difference in distribution of traces was calculated to be greater more than two standard deviations outside the mean. This work is novel, as distributed tracing has not

been previously used to detect cyber-attacks. The anomaly detector was classified as a group anomaly detection method due to the excess microservice activity of the password guessing attack.

This demonstrates that an emulated cyber-security attack targeting a microservices application can be detected using anomaly detection based on frequency distribution which supports and enhances cyber-attack detection.

The third and final research question was as follows:

3. *Can graph convolutional neural networks and traffic forecasting detect cyber attacks in microservices applications?*

The proposed method in the previous contribution is lacking as it only paid attention to the frequency of the relations between microservice calls. To address this, for the third and final contribution, it was proposed that the traffic of microservice calls to the application could be modelled using a microservice call graph. In this graph representation, the nodes represent the actual microservices and the edges represent the calls to those microservices. The actual RPC traffic of the application was also modelled using this call graph by relating the traffic to a diffusion process. This graph representation and the RPC traffic would be fed into a deep learning model in order to learn the spatial relations and temporal dependencies of the RPC traffic, and subsequently predict future traffic activity through traffic forecasting.

Like the previous contribution, the microservices application *SocialNetwork* was selected for the experiment, normal data was recorded using distributed tracing, and anomaly detection was employed to discover seeded cyber-security attacks amongst testing data. To prepare normal data for the GCNN, the application was run over an established time series, generating a data series of aggregated RPC calls per time window. For modeling the spatial relations of the application's topological architectural and learning the temporal dependencies of the RPC traffic data, a state-of-the-art GCN called the DCRNN was proposed.

To train the DCRNN model, the RPC traffic data series and the microservice call graph were fed into the model as input. Subsequently, testing data was generated in which cyber-security attacks were emulated and the trained DCRNN model was leveraged to predict RPC traffic at future time steps in the testing data sets given previously learned

regular traffic. This supported the detection of irregular RPC traffic patterns caused by the seeded cyber-attacks. To evaluate the DCRNN's ability to represent the microservice data during training, a subset of the training data was set aside as a validation data set. The training and validation values were calculated using the loss function *MAE*. The learning curves that represent these loss values were shown to converge smoothly and in close proximity to each other. This proved the DCRNN was suited for modelling the microservice data.

This approach was evaluated by conducting three experiments with the trained DCRNN in which a different cyber-attack was emulated. The following cyber-attacks were addressed in these experiments:

- a brute force password guessing attack
- a batch registration of bot accounts
- a DDoS HTTP GET Flood attack

The detection of irregular application traffic was applied by calculating the difference between the actual and predicted traffic values at run-time and calculating the mean and standard deviation. For each experiment, the prediction error results caused by the cyber-attacks were proven to exceed two standard deviations outside the mean. Like in the previous contribution, the anomaly detection technique is classified as group anomaly detection due to the volume of RPC traffic in order to emulate each of the three attacks.

These results prove that various cyber-security attacks targeting a microservices application can be detected using an threshold-based anomaly detection approach by training a graph convolutional neural network and applying traffic forecasting to perform microservice application trace prediction which supports cyber-attack detection.

## 6.2 Limitations

For the latter two contributions of our work revolving around detecting cyber-attacks targeting microservices, it is to be stressed that our distributed tracing and anomaly detection proposal has never been implemented before in existing works. Our work is

also limited in that we prepared synthetic data sets of microservices to evaluate the DCRNN model. Despite the various benefits of the MSA, there lacks an available test-bed of microservice cyber-attack activity, so there are no previous existing works to compare our proposed approaches to.

Our proposed anomaly detection for detecting cyber-attacks targeting a microservices application in last two contributions is a group-based anomaly detection method. The methodology only works with cyber-attacks which are composed of multiple RPC requests occurring, leading to a much greater quantity of RPCs over a short time span, compared to regular microservices application activity. Therefore, our proposed anomaly detection technique is not applicable for detecting cyber-attacks simulated using a small quantity of RPCs during testing. A cyber-attack that would not be detected by our method is an XSS attack which would only require a single stream of computation to upload the malicious website to the microservices environment.

## 6.3 Future Work

The first contribution of our project is concerned with predicting a subsequent event in an ongoing case and comparing two differing methodologies for implementing this objective. Another attribute that could be learned by the LSTM model is the timestamp of an event in a case as performed in recent works by [27] [129]. For work in the future, our LSTM model could be improved to predict the duration of future cases and the remainder of a potential case.

It is also worth noting that when we compare our two proposed methodologies for training our LSTM model, it has been observed that we get near similar results of accuracy when predicting a subsequent event in terms of accuracy for two different cases. In another two cases, we observe that the *Prefix* method performs slightly better than the Teacher Forcing method. We expected all results to be the same. Further work and deeper inspection is required to understand why this result.

For this research project, the DCRNN was trained only to learn the frequency of RPCs for the prediction of future traffic. The actual model only predicted the number of times a particular RPC occurred at a particular time step. This is only one attribute. Distributed tracing is also used to record the timestamp of a microservice call as well as

the duration of the call. Therefore, future work involving the DCRNN could include the the prediction of the remainder of an ongoing microservice trace. Another possibility is whether the DCRNN could also be trained to learn both attributes simultaneously and make separate predictions.

# Appendix A

# Microservices RPC Indices

Table A.1. RPCs for Brute Force Attack.

| ID | Source | Destination |
|---|---|---|
| 0000 | - | nginx-web-server + /api/user/login |
| 0001 | nginx-web-server + /api/user/login | nginx-web-server + /api/user/login |
| 0002 | nginx-web-server + /api/user/login | nginx-web-server + Login |
| 0003 | nginx-web-server + Login | user-service + Login |
| 0004 | - | user-service + Login |
| 0005 | user-service + Login | user-service + MmcGetLogin |
| 0006 | user-service + Login | user-service + MongoFindUser |
| 0007 | user-service + Login | user-service + MmcSetLogin |

Table A.2. RPCs for Batch Registration Attack.

| ID | Source | Destination |
|---|---|---|
| 0008 | - | nginx-web-server + /wrk2-api/user/register |
| 0009 | nginx-web-server + /wrk2-api/user/register | nginx-web-server + /wrk2-api/user/register |
| 0010 | nginx-web-server + /wrk2-api/user/register | nginx-web-server + RegisterUser |
| 0011 | nginx-web-server + RegisterUser | user-service + RegisterUserWithId |
| 0012 | user-service + RegisterUserWithId | user-service + MongoInsertUser |
| 0013 | user-service + RegisterUserWithId | social-graph-service + InsertUser |
| 0014 | social-graph-service + InsertUser | social-graph-service + MongoInsertUser |

Table A.3. RPCs for Distributed DoS Attack.

| ID | Source | Destination |
|---|---|---|
| 0015 | - | nginx-web-server + /wrk2-api/user-timeline/read |
| 0016 | nginx-web-server + /wrk2-api/user-timeline/read | nginx-web-server + /wrk2-api/user-timeline/read |
| 0017 | nginx-web-server + /wrk2-api /user-timeline /read | nginx-web-server + ReadUserTimeline |
| 0018 | nginx-web-server + ReadUserTimeline | user-timeline-service + ReadUserTimeline |
| 0019 | - | user-timeline-service + ReadUserTimeline |
| 0020 | user-timeline-service + ReadUserTimeline | user-timeline-service + RedisFind |
| 0021 | user-timeline-service + ReadUserTimeline | user-timeline-service + MongoFindUserTimeline |
| 0022 | user-timeline-service + ReadUserTimeline | user-timeline-service + RedisUpdate |
| 0023 | user-timeline-service + ReadUserTimeline | post-storage-service + ReadPosts |
| 0024 | post-storage-service + ReadPosts | post-storage-service + MemcachedMget |
| 0025 | post-storage-service + ReadPosts | post-storage-service + MongoFindPosts |
| 0026 | post-storage-service + ReadPosts | post-storage-service + MmcSetPost |

Table A.4. RPCs for Regular Traffic.

| ID | Source | Destination |
|---|---|---|
| 0027 | - | nginx-web-server + /wrk2-api/post/compose |
| 0028 | nginx-web-server + /wrk2-api/post/compose | nginx-web-server + /wrk2-api/post/compose |
| 0029 | nginx-web-server + /wrk2-api/post/compose | nginx-web-server + ComposePost |
| 0030 | nginx-web-server + ComposePost | text-service + UploadText |
| 0031 | nginx-web-server + ComposePost | media-service + UploadMedia |
| 0032 | nginx-web-server + ComposePost | user-service + UploadUserWithUserId |
| 0033 | nginx-web-server + ComposePost | unique-id-service + UploadUniqueId |
| 0034 | text-service + UploadText | user-mention-service + UploadUserMentions |
| 0035 | text-service + UploadText | url-shorten-service + UploadUrls |
| 0036 | text-service + UploadText | compose-post-service + UploadText |
| 0037 | media-service + UploadMedia | compose-post-service + UploadMedia |
| 0038 | user-service + UploadUserWithUserId | compose-post-service + UploadCreator |
| 0039 | compose-post-service + UploadMedia | compose-post-service + RedisHashSet |
| 0040 | compose-post-service + UploadCreator | compose-post-service + RedisHashSet |
| 0041 | user-mention-service + UploadUserMentions | compose-post-service + UploadUserMentions |
| 0042 | url-shorten-service + UploadUrls | compose-post-service + UploadUrls |
| 0043 | compose-post-service + UploadUrls | compose-post-service + RedisHashSet |
| 0044 | compose-post-service + UploadUserMentions | compose-post-service + RedisHashSet |

| | | |
|---|---|---|
| 0045 | compose-post-service + UploadUser-Mentions | post-storage-service + StorePost |
| 0046 | compose-post-service + UploadUser-Mentions | user-timeline-service + WriteUser-Timeline |
| 0047 | compose-post-service + UploadUser-Mentions | write-home-timeline-service + FanoutHomeTimelines |
| 0048 | unique-id-service + UploadUniqueId | compose-post-service + Upload-UniqueId |
| 0049 | compose-post-service + Upload-UniqueId | compose-post-service + RedisHash-Set |
| 0050 | compose-post-service + UploadText | compose-post-service + RedisHash-Set |
| 0051 | compose-post-service + UploadText | write-home-timeline-service + FanoutHomeTimelines |
| 0052 | compose-post-service + UploadText | post-storage-service + StorePost |
| 0053 | compose-post-service + UploadText | user-timeline-service + WriteUser-Timeline |
| 0054 | write-home-timeline-service + FanoutHomeTimelines | social-graph-service + GetFollowers |
| 0055 | write-home-timeline-service + FanoutHomeTimelines | write-home-timeline-service + Redis-Update |
| 0056 | post-storage-service + StorePost | post-storage-service + MongoInsert-Post |
| 0057 | social-graph-service + GetFollowers | social-graph-service + RedisGet |
| 0058 | social-graph-service + GetFollowers | social-graph-service + MongoFind-User |
| 0059 | social-graph-service + GetFollowers | social-graph-service + RedisInsert |
| 0060 | user-timeline-service + WriteUser-Timeline | user-timeline-service + MongoFind-User |
| 0061 | user-timeline-service + WriteUser-Timeline | user-timeline-service + MongoInsert |
| 0062 | user-timeline-service + WriteUser-Timeline | user-timeline-service + RedisUpdate |

# Bibliography


[1] L Dhanabal and SP Shantharajah. A study on nsl-kdd dataset for intrusion detection system based on classification algorithms. *International journal of advanced research in computer and communication engineering*, 4(6):446–452, 2015.

[2] Konark Truptiben Dave. Brute-force attack 'seeking but distressing'. *Int. J. Innov. Eng. Technol. Brute-force*, 2(3):75–78, 2013.

[3] Jeff Petters. What is a brute force attack? https://www.varonis.com/blog/brute-force-attack, December 2021. (Accessed on 03/23/2022).

[4] Ved Prakash Singh and Preet Pal. Survey of different types of captcha. *International Journal of Computer Science and Information Technologies*, 5(2):2242–2245, 2014.

[5] V Revuelto, S Meintanis, and K Socha. Ddos overview and response guide. https://cert.europa.eu/static/WhitePapers/CERT-EU_Security_Whitepaper_DDoS_17-003.pdf, March 2017. (Accessed on 04/06/2022).

[6] Net Scout. What is a syn flood ddos attack? — f5. https://www.f5.com/services/resources/glossary/syn-flood#:~:text=A%20SYN%20flood%2C%20also%20known,overwhelm%20it%20with%20open%20connections., 2022. (Accessed on 04/07/2022).

[7] Imperva. What is a tcp syn flood — ddos attack glossary — imperva. https://www.imperva.com/learn/ddos/syn-flood/, 2021. (Accessed on 04/07/2022).

[8] Imperva. What is snmp reflection and amplification — ddos attack glossary — imperva. https://www.imperva.com/learn/ddos/snmp-reflection/, 2021. (Accessed on 04/07/2022).

[9] Inc Cloudflare. Http flood ddos attack — cloudflare. https://www.cloudflare.com/learning/ddos/http-flood-ddos-attack/, 2022. (Accessed on 04/07/2022).

[10] Maryam M Najafabadi, Taghi M Khoshgoftaar, Chad Calvert, and Clifford Kemp. User behavior anomaly detection for application layer ddos attacks. In *2017 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 154–161. IEEE, 2017.

[11] Jiyu Chen, Heqing Huang, and Hao Chen. Informer: Irregular traffic detection for containerized microservices rpc in the real world. *High-Confidence Computing*, page 100050, 2022.

[12] Signal Sciences. What are bot attacks? bot mitigation for web apps & apis. https://www.signalsciences.com/glossary/bot-attack-protection/, 2020. (Accessed on 04/06/2022).

[13] Avanish Pathak. An analysis of various tools, methods and systems to generate fake accounts for social media. *Northeastern University Boston, Massachusetts December*, 2014.

[14] Charlie Belmer. A nosql injection primer (with mongodb). https://nullsweep.com/a-nosql-injection-primer-with-mongo/, September 2020. (Accessed on 03/23/2022).

[15] Ahmed M Eassa, Mohamed Elhoseny, Hazem M El-Bakry, and Ahmed S Salama. Nosql injection attack detection in web applications using restful service. *Programming and Computer Software*, 44(6):435–444, 2018.

[16] Invicti. What is nosql injection and how can you prevent it? — invicti. https://www.invicti.com/blog/web-security/what-is-nosql-injection/, 2022. (Accessed on 04/07/2022).

[17] Panda Security Mediacenter. What is a man-in-the-middle (mitm) attack? definition and prevention - panda security mediacenter. https://www.pandasecurity.com/en/mediacenter/security/man-in-the-middle-attack/, 2021. (Accessed on 04/07/2022).

[18] Boudewijn F Van Dongen, Ana Karla A de Medeiros, HMW Verbeek, AJMM Weijters, and Wil MP van Der Aalst. The prom framework: A new era in process mining tool support. In *International conference on application and theory of petri nets*, pages 444–454. Springer, 2005.

[19] Wil MP Van der Aalst and Ana Karla A de Medeiros. Process mining and security: Detecting anomalous process executions and checking process conformance. *Electronic Notes in Theoretical Computer Science*, 121:3–21, 2005.

[20] Sean Carlisto de Alvarenga, B Zarpel, and R Miani. Discovering attack strategies using process mining. In *Proc. of the 11th Advanced Int'l Conf. on Telecommunications*, pages 119–125, 2015.

[21] Sebastian Mauser and Tobias Eggendorfer. Detecting security attacks by process mining. *Algorithms and Tools for Petri Nets*, page 33, 2017.

[22] Sean Carlisto De Alvarenga, Sylvio Barbon Jr, Rodrigo Sanches Miani, Michel Cukier, and Bruno Bogaz Zarpelão. Process mining and hierarchical clustering to help intrusion alert visualization. *Computers & Security*, 73:474–491, 2018.

[23] Wil Van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE transactions on knowledge and data engineering*, 16(9):1128–1142, 2004.

[24] Larry R Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5:64–67, 2001.

[25] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5528–5531. IEEE, 2011.

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

[27] Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. Predictive business process monitoring with lstm neural networks. In *International Conference on Advanced Information Systems Engineering*, pages 477–492. Springer, 2017.

[28] Mirko Polato, Alessandro Sperduti, Andrea Burattin, and Massimiliano de Leoni. Time and activity sequence prediction of business process instances. *Computing*, 100(9):1005–1031, 2018.

[29] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. Predicting process behaviour using deep learning. *Decision Support Systems*, 100:129–140, 2017.

[30] Dominic Breuker, Martin Matzner, Patrick Delfmann, and Jörg Becker. Comprehensible predictive models for business processes. *Mis Quarterly*, 40(4):1009–1034, 2016.

[31] Andrea Vedaldi and Karel Lenc. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 689–692, 2015.

[32] Ruoyu Li and Junzhou Huang. Learning graph while training: An evolving graph convolutional neural network. *arXiv preprint arXiv:1708.04675*, 2017.

[33] James Atwood and Don Towsley. Diffusion-convolutional neural networks. *Advances in neural information processing systems*, 29, 2016.

[34] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. *arXiv preprint arXiv:1908.01000*, 2019.

[35] Nurul A Asif, Yeahia Sarker, Ripon K Chakrabortty, Michael J Ryan, Md Hafiz Ahamed, Dip K Saha, Faisal R Badal, Sajal K Das, Md Firoz Ali, Sumaya I Moyeen, et al. Graph neural network: A comprehensive review on non-euclidean space. *IEEE Access*, 2021.

[36] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*, 28, 2015.

[37] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*, 2017.

[38] Nanyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, and Wen-tau Yih. Cross-sentence n-ary relation extraction with graph lstms. *Transactions of the Association for Computational Linguistics*, 5:101–115, 2017.

[39] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):1–23, 2019.

[40] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[41] Tomasz Danel, Przemysław Spurek, Jacek Tabor, Marek Śmieja, Łukasz Struski, Agnieszka Słowik, and Łukasz Maziarka. Spatial graph convolutional networks. In *International Conference on Neural Information Processing*, pages 668–675. Springer, 2020.

[42] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations (ICLR '18)*, 2018.

[43] Davide Andreoletti, Sebastian Troia, Francesco Musumeci, Silvia Giordano, Guido Maier, and Massimo Tornatore. Network traffic prediction based on diffusion convolutional recurrent neural networks. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 246–251. IEEE, 2019.

[44] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[45] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, et al. An open-source benchmark suite for cloud and iot microservices. *arXiv preprint arXiv:1905.11055*, 2019.

[46] Yu Gan and Christina Delimitrou. The architectural implications of cloud microservices. *IEEE Computer Architecture Letters*, 17(2):155–158, 2018.

[47] Yuqiong Sun, Susanta Nanda, and Trent Jaeger. Security-as-a-service for microservices-based cloud applications. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 50–57. IEEE, 2015.

[48] Antonio Nehme, Vitor Jesus, Khaled Mahbub, and Ali Abdallah. Securing microservices. *IT Professional*, 21(1):42–49, 2019.

[49] Sam Neuman. Building microservices: Designing fine-grained systems. *Oreilly & Associates Inc*, 2015.

[50] Dongjin Yu, Yike Jin, Yuqun Zhang, and Xi Zheng. A survey on security issues in services communication of microservices-enabled fog applications. *Concurrency and Computation: Practice and Experience*, 31(22):e4436, 2019.

[51] AR Manu, Jitendra Kumar Patel, Shakil Akhtar, VK Agrawal, and KN Bala Subramanya Murthy. A study, analysis and deep dive on cloud paas security in terms of docker container security. In *2016 international conference on circuit, power and computing technologies (ICCPCT)*, pages 1–13. IEEE, 2016.

[52] Guojun Wang, Qin Liu, and Jie Wu. Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 735–737, 2010.

[53] Aliyu Lawal Aliyu, Peter Bull, and Ali Abdallah. A trust management framework for network applications within an sdn environment. In *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 93–98. IEEE, 2017.

[54] Wanpeng Li and Chris J Mitchell. Analysing the security of google's implementation of openid connect. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 357–376. Springer, 2016.

[55] Andre Bento, Jaime Correia, Ricardo Filipe, Filipe Araujo, and Jorge Cardoso. Automated analysis of distributed tracing: Challenges and research directions. *Journal of Grid Computing*, 19(1):1–15, 2021.

[56] Maria C Borges, Sebastian Werner, and Ahmet Kilic. Faaster troubleshooting-evaluating distributed tracing approaches for serverless applications. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*, pages 83–90. IEEE, 2021.

[57] Sina Niedermaier, Falko Koetter, Andreas Freymann, and Stefan Wagner. On observability and monitoring of distributed systems–an industry interview study. In *International Conference on Service-Oriented Computing*, pages 36–52. Springer, 2019.

[58] Raghavendra Chalapathy, Edward Toth, and Sanjay Chawla. Group anomaly detection using deep generative models. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 173–189. Springer, 2018.

[59] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[60] Naoki Abe, Bianca Zadrozny, and John Langford. Outlier detection by active learning. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 504–509, 2006.

[61] Ingo Steinwart, Don Hush, and Clint Scovel. A classification framework for anomaly detection. *Journal of Machine Learning Research*, 6(2), 2005.

[62] Fábio Bezerra, Jacques Wainer, and Wil MP van der Aalst. Anomaly detection using process mining. In *Enterprise, business-process and information systems modeling*, pages 149–161. Springer, 2009.

[63] Sasho Nedelkoski, Jorge Cardoso, and Odej Kao. Anomaly detection and classification using distributed tracing and deep learning. In *2019 19th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGRID)*, pages 241–250. IEEE, 2019.

[64] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[65] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Data mining introduction. *People's Posts and Telecommunications Publishing House, Beijing*, 2006.

[66] Shyam Boriah, Varun Chandola, and Vipin Kumar. Similarity measures for categorical data: A comparative evaluation. In *Proceedings of the 2008 SIAM international conference on data mining*, pages 243–254. SIAM, 2008.

[67] Marc M Van Hulle. Self-organizing maps., 2012.

[68] Khaled Labib and Rao Vemuri. Nsom: A real-time network-based intrusion detection system using self-organizing maps. *Networks and Security*, 21(1), 2002.

[69] Patrick L Brockett, Xiaohua Xia, and Richard A Derrig. Using kohonen's self-organizing feature map to uncover automobile bodily injury claims fraud. *Journal of Risk and Insurance*, pages 245–274, 1998.

[70] Konstantinos Drossos, Shayan Gharib, Paul Magron, and Tuomas Virtanen. Language modelling for sound event detection with teacher forcing and scheduled sampling. *arXiv preprint arXiv:1907.08506*, 2019.

[71] HMW Verbeek, JCAM Buijs, BF Van Dongen, and Wil MP van der Aalst. Prom 6: The process mining toolkit. *Proc. of BPM Demonstration Track*, 615:34–39, 2010.

[72] Process and Data Science Group RWTH Aachen University. Process mining. http://www.processmining.org/, 2021. (Accessed on 04/11/2022).

[73] PROM Tools. exercises:start — prom tools. https://promtools.org/doku.php?id=exercises:start, 2021. (Accessed on 04/13/2022).

[74] Boudewijn F Van Dongen, AK Alves de Medeiros, and Lijie Wen. Process mining: Overview and outlook of petri net discovery algorithms. *transactions on petri nets and other models of concurrency II*, pages 225–242, 2009.

[75] Sander JJ Leemans, Dirk Fahland, and Wil MP Van Der Aalst. Discovering block-structured process models from event logs containing infrequent behaviour. In *International conference on business process management*, pages 66–78. Springer, 2013.

[76] Lincoln Laboratory. Mit lincoln laboratory: Darpa intrusion detection evaluation. https://archive.ll.mit.edu/ideval/data/2000/LLS_DDOS_1.0.html, 2000. (Accessed on 04/13/2022).

[77] Bin Zhu and Ali A Ghorbani. Alert correlation for extracting attack strategies. *Int. J. Netw. Secur.*, 3(3):244–258, 2006.

[78] Wil Van Der Aalst. Process mining. *Communications of the ACM*, 55(8):76–83, 2012.

[79] Hamid Farhadi, Maryam AmirHaeri, and Mohammad Khansari. Alert correlation and prediction using data mining and hmm. *The ISC International Journal of Information Security*, 3(2):77–101, 2011.

[80] Mahdiyeh Barzegar and Mehdi Shajari. Attack scenario reconstruction using intrusion semantics. *Expert Systems with Applications*, 108:119–133, 2018.

[81] Stephen Jacob, Yuansong Qiao, Paul Jacob, and Brian Lee. Using recurrent neural networks to predict future events in a case with application to cyber security. In *BUSTECH*, pages 13–19, 2020.

[82] Antonio Gulli and Sujit Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[83] Boudewijn van Dongen. Bpi challenge 2012. https://data.4tu.nl/articles/dataset/BPI_Challenge_2012/12689204, 2020. (Accessed on 03/21/2022).

[84] RP Bose and Wil MP van der Aalst. Process mining applied to the bpi challenge 2012: Divide and conquer while discerning resources. In *International Conference on Business Process Management*, pages 221–222. Springer, 2012.

[85] Ward Steeman. Bpi challenge 2013, incidents. https://data.4tu.nl/articles/dataset/BPI_Challenge_2013_incidents/12693914/1, 2020. (Accessed on 03/21/2022).

[86] Nijat Mehdiyev, Joerg Evermann, and Peter Fettke. A multi-stage deep learning approach for business process event prediction. In *2017 IEEE 19th conference on business informatics (CBI)*, volume 1, pages 119–128. IEEE, 2017.

[87] Bpi challenge 2014. https://data.4tu.nl/collections/_/5065469/1, 2020. (Accessed on 03/22/2022).

[88] Alex M Lamb, Anirudh Goyal ALIAS PARTH GOYAL, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. *Advances in neural information processing systems*, 29, 2016.

[89] Brian Lee and Stephen Jacob. Bpic_2012 · stephensbranch · sri-ait-ie / phd-projects / stephenj · gitlab. https://gitlab.com/sri-ait-ie/phd-projects/stephenj/-/tree/stephensBranch/BPIC_2012, 2022. (Accessed on 04/08/2022).

[90] Brian Lee and Stephen Jacob. Bpic_2013 · stephensbranch · sri-ait-ie / phd-projects / stephenj · gitlab. https://gitlab.com/sri-ait-ie/phd-projects/stephenj/-/tree/stephensBranch/BPIC_2013, 2022. (Accessed on 04/08/2022).

[91] Brian Lee and Stephen Jacob. Bpic_2014 · stephensbranch · sri-ait-ie / phd-projects / stephenj · gitlab. https://gitlab.com/sri-ait-ie/phd-projects/stephenj/-/tree/stephensBranch/BPIC_2014, 2022. (Accessed on 04/08/2022).

[92] Brian Lee and Stephen Jacob. helpdesk_data_set · stephensbranch · sri-ait-ie / phd-projects / stephenj · gitlab. https://gitlab.com/sri-ait-ie/phd-projects/stephenj/-/tree/stephensBranch/helpDesk_Data_Set, 2022. (Accessed on 04/08/2022).

[93] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[95] Tetiana Yarygina and Anya Helene Bagge. Overcoming security challenges in microservice architectures. In *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 11–20. IEEE, 2018.

[96] Abdelhakim Hannousse and Salima Yahiouche. Securing microservices and microservice architectures: A systematic mapping study. *Computer Science Review*, 41:100415, 2021.

[97] Deathstarbench/socialnetwork at master · delimitrou/deathstarbench · github. https://github.com/delimitrou/DeathStarBench/tree/master/socialNetwork, 2022. (Accessed on 03/14/2022).

[98] APLOS. Microservices. http://microservices.ece.cornell.edu/#contact, April 2021. (Accessed on 03/14/2022).

[99] Frequency distribution - quick introduction. https://www.spss-tutorials.com/frequency-distribution-what-is-it/#:~:text=A%20frequency%20distribution%20is%20an,used%20for%20summarizing%20categorical%20variables., 2022. (Accessed on 03/14/2022).

[100] Nginx. https://nginx.org/en/, 2020. (Accessed on 03/14/2022).

[101] Apache Software Foundation. Apache thrift - home. https://thrift.apache.org/, 2022. (Accessed on 03/14/2022).

[102] The Jaeger Authors. Jaeger: open source, end-to-end distributed tracing. https://www.jaegertracing.io/, 2022. (Accessed on 03/15/2022).

[103] Yang Wang. Github - elastic/elasticsearch: Free and open, distributed, restful search engine. https://github.com/elastic/elasticsearch, 2020. (Accessed on 04/01/2022).

[104] Brian Lee and Stephen Jacob. socialnetwork_v2 · socialnetworkmsabranch · sri-ait-ie / phd-projects / stephenj · gitlab. https://gitlab.com/sri-ait-ie/phd-projects/stephenj/-/tree/socialNetworkMSABranch/socialNetwork_V2, 2022. (Accessed on 04/28/2022).

[105] Stephen Jacob, Yuansong Qiao, and Brian A Lee. Detecting cyber security attacks against a microservices application using distributed tracing. In *ICISSP*, pages 588–595, 2021.

[106] Brian Lee and Stephen Jacob. socialnetwork_cyberattackdata · socialnetworkmsabranch · sri-ait-ie / phd-projects / stephenj · gitlab. https://gitlab.com/sri-ait-ie/phd-projects/stephenj/-/tree/socialNetworkMSABranch/socialNetwork_CyberAttackData, 2022. (Accessed on 05/04/2022).

[107] Mongodb: The application data platform — mongodb. https://www.mongodb.com/, 2020. (Accessed on 03/14/2022).

[108] Brad Fitzpatrick. Distributed caching with memcached. *Linux journal*, 124, 2004.

[109] Sacha Brostoff and M Angela Sasse. "ten strikes and you're out": Increasing the number of login attempts can improve password usability. *Human-Computer Interation, Security*, 2003.

[110] Do Quoc Le, Taeyoel Jeong, H Eduardo Roman, and James Won-Ki Hong. Traffic dispersion graph based anomaly detection. In *Proceedings of the Second Symposium on Information and Communication Technology*, pages 36–41, 2011.

[111] François-Xavier Aubet, Marc-Oliver Pahl, Stefan Liebald, and Mohammad Reza Norouzian. Graph-based anomaly detection for iot microservices. *Measurements*, 120(140):160, 2018.

[112] U Sekar. Applications of graph theory in computer science. *International Journal of Electronics Communication and Computer Engineering*, 4:2278–4209, 2013.

[113] Eleni I Vlahogianni, Matthew G Karlaftis, and John C Golias. Short-term traffic forecasting: Where we are and where we're going. *Transportation Research Part C: Emerging Technologies*, 43:3–19, 2014.

[114] Brian Lee and Stephen Jacob. mymicroservicerpcproject · journal_branch · sri-ait-ie / phd-projects / stephenj · gitlab. https://gitlab.com/sri-ait-ie/phd-projects/stephenj/-/tree/journal_Branch/myMicroserviceRPCProject, 2022. (Accessed on 05/04/2022).

[115] Tanwi Mallick, Prasanna Balaprakash, Eric Rask, and Jane Macfarlane. Graph-partitioning-based diffusion convolutional recurrent neural network for large-scale traffic forecasting. *Transportation Research Record*, 2674(9):473–488, 2020.

[116] Shang-Hua Teng. Scalable algorithms for data and network analysis. *Foundations and Trends® in Theoretical Computer Science*, 12(1–2):1–274, 2016.

[117] Michael C Fu. Stochastic gradient estimation. *Handbook of simulation optimization*, pages 105–147, 2015.

[118] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[119] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[120] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in neural information processing systems*, 28, 2015.

[121] Stephen Jacob, Yuansong Qiao, Yuhang Ye, and Brian Lee. Anomalous distributed traffic: Detecting cyber security attacks amongst microservices using graph convolutional networks. *Computers & Security*, page 102728, 2022.

[122] Brian Lee and Stephen Jacob. socialnetwork · journal_branch · sri-ait-ie / phd-projects / stephenj · gitlab. https://gitlab.com/sri-ait-ie/phd-projects/stephenj/-/tree/journal_Branch/socialNetwork, 2022. (Accessed on 05/04/2022).

[123] Jeff Peters. What is a brute force attack? https://www.varonis.com/blog/brute-force-attack, 2020. (Accessed on 04/04/2022).

[124] Imperva. What does ddos mean? — distributed denial of service explained — imperva. https://www.imperva.com/learn/ddos/denial-of-service/?utm_campaign=Incapsula-moved, 2021. (Accessed on 04/05/2022).

[125] Huan Xu, Constantine Caramanis, and Shie Mannor. Robustness and regularization of support vector machines. *Journal of machine learning research*, 10(7), 2009.

[126] Giovanni Apruzzese, Michele Colajanni, Luca Ferretti, and Mirco Marchetti. Addressing adversarial attacks against security systems based on machine learning. In *2019 11th international conference on cyber conflict (CyCon)*, volume 900, pages 1–18. IEEE, 2019.

[127] Anthony D Joseph, Pavel Laskov, Fabio Roli, J Doug Tygar, and Blaine Nelson. Machine learning methods for computer security (dagstuhl perspectives workshop 12371). In *Dagstuhl Manifestos*, volume 3, pages 1–30. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[128] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and J Doug Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58, 2011.

[129] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, Puneet Agarwal, et al. Long short term memory networks for anomaly detection in time series. In *Proceedings*, volume 89, pages 89–94, 2015.