Computers & Security 118 (2022) 102728

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Anomalous distributed traffic: Detecting cyber security attacks amongst microservices using graph convolutional networks

Stephen Jacob*, Yuansong Qiao, Yuhang Ye, Brian Lee

Technological University of the Shannon: Midlands Midwest, Dublin Road, Athlone, Co. Westmeath, Ireland

ARTICLE INFO

Article history:
Received 5 November 2021
Revised 10 April 2022
Accepted 15 April 2022
Available online 22 April 2022

Keywords:
Cyber security
Microservices
Distributed tracing
Anomaly detection
Graph convolutional network
Traffic forecasting

ABSTRACT

Currently, microservices are trending as the most popular software application design architecture. Software organisations are also being targeted by more cyber-attacks every day and newer security measures are in high demand. One available measure is the application of anomaly detection, which is defined as the discovery of irregular or unusual activity that occurs to a greater or lesser degree than normal occurrences in a data series. In this paper, we continue existing work where various real-world cyber-attacks are executed against a running microservices application, and the application traffic is logged and returned in the form of distributed traces. A Diffusion Convolutional Recurrent Neural Network is used to model the set of distributed traces and learn the spatial and temporal dependencies of the application traffic. Subsequently, the model is used to make predictions for ongoing microservice activity and threshold-based anomaly detection is applied to detect irregular microservice activity indicating the presence of seeded cyber security attacks, or anomalies. The cyber-attacks used to evaluate this approach include a brute force attack, a batch registration of bot accounts and a distributed denial of service attack.

1. Introduction

Cyber security is currently one of the more significant problems across the world. Every day, hackers are targeting more software organizations with a variety of well-defined cyber-attacks. In recent years, the microservices software architecture has been implemented by many popular software application brands, including Twitter, Amazon, Netflix and PayPal (Gan et al., 2019b). Consequently, cyber security personnel overseeing these applications require more up-to-date means of detecting the cyber assaults injected into their application model.

In our previous work Jacob et al. (2021), we investigated cyber-attacks targeting a microservices application by monitoring the overall behaviour of the application using distributed tracing and detected the anomalous activity of a cyber-attack by calculating the frequency distribution of unique traces. Such distributed traces capture and record the sequence of API calls between the components of a distributed application as a microservice call graph where the nodes of the graph are the actual microservices and the edges represent calls to microservices. A sequence of such call graphs over time captures the spatio-temporal characteristics from the API call traffic of a microservice application. Graph based anomaly detection is then used to look for variations in the application call traffic that indicates unusual or abnormal behaviour.

In this work, we propose to build on our earlier efforts using anomaly detection to detect cyber-attacks in microservice traffic by exploring the application of graph based anomaly detection to API call traffic graphs produced by the microservices application. Specifically, we use the microservice call graph and data to train a graph convolutional neural network (GCNN) to capture the existing spatial and temporal dynamics within the tracing data. By using a GCNN to model the application topology and predict ongoing traffic, the irregular microservice traffic caused by various seeded cyber-attacks is detected.

In this paper, we use a distributed tracing tool to monitor a microservices application with the goal of detecting cyber security attacks targeting the application. We also define a Diffusion Convolutional Recurrent Neural Network (DCRNN), a state-of-the-art GCNN designed to learn the directional behaviour of the traffic modelled on a directed graph and subsequently perform traffic forecasting for future time steps. In our experiment, we run a microservice application and simulate different cyber security attacks. We detect these attacks by leveraging the DCRNN model to discover the irregular microservice traffic caused as a result of said attacks.

A user's request to an application produces a sequence of related microservice calls. This sequence of remote procedure calls (RPC)s is logged using distributed tracing. Regular user calls made

* Corresponding author.
E-mail address: s.jacob@research.ait.ie (S. Jacob).

https://doi.org/10.1016/j.cose.2022.102728

to the application results in a set of distributed traces comprised of these RPCs. The DCRNN model is trained to learn from this RPC traffic and discover the spatial relations and temporal dynamics. This approach is used to determine the presence of RPC dynamics in a fixed time window that do not conform to the regular behaviour of normal microservice application traffic. The aim of this work is to detect anomalies by comparing the computed RPC traffic related to a cyber security attack against the RPC traffic from a normal data set.

It should be noted that our work to perform graph-based anomaly detection is loosely similar to the approach used by the authors of Chen et al. (2019). The novelty of our approach is to provide a more simplified process of training a GCNN model and learning the spatio-temporal dynamics of RPC traffic. In our approach, we train only a single model to learn the entire microservice application as opposed to Chen et al. (2019) in which multiple models are trained to each learn a different subsystem of the application.

Our main contributions in this paper are summarized as follows:

- We use a directed graph to model the entire polylithic architecture of a microservices application and the inter-relations between the individual services. Using this graph, the application traffic from one microservice node to its neighbouring nodes can be related to a **diffusion process**.
- We propose the **Diffusion Convolution Recurrent Neural Network** to learn the spatial and temporal dependencies of the application traffic over a time series using a diffusion convolution operation.
- We study the traffic forecasting problem to predict microservice traffic at a future time step given previously learned traffic.
- We apply anomaly detection to discover cyber security attacks injected into microservice traffic by measuring the irregularity of the RPCs made as a result of the cyber security attacks.

This paper is structured as follows: Section 2 outlines the related literature works. Section 3 presents background information on the microservices architecture, the fields of distributed tracing and anomaly detection. Section 4 presents an overview of our proposal and present a high-level description of our approach. Section 5 describes the microservices application we selected for our experiment, the different cyber security attacks investigated and simulated against the application, and outlines the application of anomaly detection to detect the cyber security attacks seeded amongst the application's traffic flow. In Section 6, we discuss the advantages and limitations of our proposed approach and Section 7 provides a conclusion to our proposal and possible future work.

2. Related works

This section presents a literature review of related works, including a number of machine learning-based approaches to perform graph-based anomaly detection on network traffic.

Deep neural networks (DNN) have been used to model data and discover the underlying behaviour in the data. A particular class of neural networks, a recurrent neural network (RNN) is used to model sequential data. Such a data series is usually represented as a case, a sequence of process events. Tax et al. (2017) used a particular type of RNN, the Long Short Term Memory (LSTM) neural network Hochreiter and Schmidhuber (1997) which detects long or short term dependencies in cases. This LSTM-based framework was used to learn the typical form of cases and subsequently predict future events and the timestamp of said events. The performance of this framework was evaluated by training and learning the behaviour of logged cases from two available data sets, and the results were shown to outperform a previous methodology by Polato et al. (2018).

Deep learning models have also been used to model the traffic flow of a network domain. A Convolutional Neural Networks (CNN) is a DNN suited for modeling and analyzing graphs constructs and imagery. The CNN model would observe and learn the spatial relations of the traffic flow. The authors for Ma et al. (2017) proposed a CNN model to learn the network traffic as images to capture the spatial and temporal dynamics of the data and predict the network traffic speed. This CNN algorithm was tested using two data sets composed of real-world transportation traffic. The CNN-based framework's performance was evaluated against four prevailing statistical algorithms and three deep learning-based models and was shown to outperform these models with an improved accuracy of 42.91%.

Wu and Tan (2016) proposed a deep model with a CNN and LSTM combined architecture (CLTFP) where the CNN component was used to capture and learn the spatial features of the traffic flow while the LSTM component was used to learn the temporal dependencies. Afterwards, the trained model was used to perform short term traffic forecasting. The predictions returned by the CLTFP model were then compared with those of other models including an LSTM, a shallow neural network and a stacked autoencoder model Lv et al. (2014), and the CLTFP model was shown to outperform the other models in terms of prediction accuracy and spatial distribution.

In recent works, graph neural networks (GNNs) have become popular for modelling nodes and dependencies found in various domains including life science and social networks Kung-Hsiang (2019). A variant of GNNs called *Spatial-Temporal GNNs* (STGNN)s aim to capture the spatial and temporal features within correlated data graphs simultaneously to predict future activity in a wide range of applications Wu et al. (2020). The work by Yu et al. (2017) highlights that timely traffic forecasting is essential for safe traffic control and that traditional mathematical approaches like linear regression are not suited for future long-term traffic prediction. A STGNN was proposed to model the time-series based prediction problem of a traffic domain. The network of road segments were modelled on graphs using convolution structures to enable fast training with the STGNN and extract the spatial and temporal features. This approach was evaluated in a series of experiments using various real-world traffic data sets as examples and results show that the model converges easily and outperforms state-of-the-art baseline models.

A recent GCNN, known as the Diffusion Convolutional Recurrent Neural Network (DCRNN) is a state-of-the-art model designed for learning the complex spatial and temporal features in traffic flow. The authors for Li et al. (2017) outlined the application of spatio-temporal traffic forecasting in the domain of road networks. They proposed that the traffic be modelled as an active diffusion process on a directed graph. After learning the ground truth observations, predictions of future traffic activity are generated. This methodology was tested using two different databases containing real-world road network traffic. The first data set contains traffic data derived from 207 sensors throughout Los Angeles County over a period of four months. This framework was tested, and was proven to outperform baseline state-of-the-art frameworks by a margin of 12% to 15%.

The work for this paper is similar to that of Chen et al. (2019) to use a GCNN to detect irregular real-world RPC traffic. The latter aimed to discover cyber security issues within the thousands of RPCs resulting from numerous microservices. In this work, a two-step process was performed to trace and log the RPC traffic and detect anomalies. First, the logged RPC traffic from active microservice functionality was analyzed and correlating RPC chain patterns in the data were identified using a density-clustering al-

gorithm. These chain patterns represent a subsystem of the overall microservice functionality. A GCNN is then used to model each subsystem of the RPC traffic and learn the spatio-temporal dependencies of the traffic to solve the irregular RPC prediction problem. Using these GCNNs, a series of individual predictions can be made for each pre-existing subsystems. This approach was evaluated using two case studies composed of real-world malicious traffic threat models including a batch registration of bot accounts and account cracking.

2.1. Comparable works

The authors for Le et al. (2011) use a traffic dispersion graph methodology to model network traffic over time. This approach is composed of two parts: one that learns the static properties of the graph and a dynamic aspect that models the dependencies of the temporal dynamics of the TDGs as a function of time. Anomalous traffic is defined as the traffic caused by different forms of illegal computing behaviour, including DDoS attacks, scanning and Internet worms. This TDG model was used to detect anomalies via irregular network traffic occurring over time, as well as to determine the causes of such anomalies. This TDG method was evaluated using two data sets of traffic traces and was able to detect a cyber-attack with 100% accuracy.

Yao et al. (2019) proposes a high-level attack detection framework for network communication data by using a hybrid CNN/LSTM deep learning model called STDeepGraph to learn high-level representations of network flow traffic. This work uses a temporal communication graph to model the network communication structure and a distance graph kernel to map the communication into a high-dimensional space. The CNN component was used for extracting the spatial features of the network flow and the LSTM for the temporal features. Finally, the model uses a softmax classification function to classify the network traffic as benign or malicious. Two experiments were performed to evaluate the STDeepGraph using real-world network attack data sets with various attacks seeded amongst the traffic flow. The model's performance was evaluated using various metrics including accuracy. The results show that this method outperforms baseline methods in terms of accuracy and loss.

The work by Lee et al. (2020) proposes a deep learning model that takes a graph representation of traffic-based data transforms over time, and learns the spatio-temporal dynamics of the data. The model was used to predict the dynamic anomalies by measuring the non-Euclidean distance between the actual values and the output predictions. This was done by computing the affinity score of an existing data entity. Subsequently, a threshold value is established to detect anomalous behaviour. This approach was evaluated the using two available traffic-related data sets of network traffic and public transport traffic. The metrics used to evaluate the model were the sum of absolute differences for the affinity score prediction and accuracy for the prediction of existing connections. The model was shown to have competitive results that were comparable to state-of-the art techniques.

3. Background information

This section describes background information on the microservices architecture, the process of distributed tracing and the field of anomaly detection.

3.1. Microservices

The microservice architecture (MSA), or microservices, is a service-oriented software architectural design where the application is decoupled into several smaller inter-connected services. Each *micro*service handles one specific business function of the application's overall functionality such as a new user registering or a database query. In a microservices application, a single microservice is a well-defined interface that operates alongside other microservices but can be developed, tested, scaled and deployed independently due to the application's polylithic design. This interface can be called in response to a user's RESTful API call or an RPC Sun et al. (2015).

3.2. Distributed tracing

The process of distributed tracing is defined as the capability to log and monitor the process workflow propagating throughout a cloud-native distributed system at run-time. In a microservices application, a user's HTTP request typically requires multiple microservices resulting in a sequence of operations. This set of microservices is then recorded as a **distributed trace**, a detailed log of the execution path throughout the application. A single trace is composed of units known as **spans** which share a **traceID**. A recorded span represents a single microservice operation executed in response to a user's HTTP request and sports a unique **spanID**. Characteristics recorded in the span include the name, timestamp and the duration of the microservice operation being called.

3.3. Anomaly detection

Anomaly detection is defined as the discovery of irregular behaviour or instances within a data set Anodot (2020). These anomalous instances, or outliers, either do not conform to the majority of the instances in the data set or appear at a greater or lesser frequency. Real-world examples of anomalies include enemy activity detected by military surveillance, ailments displayed by medical imaging and the presence of cyber-attacks within a computer system. Traditional anomaly detection is described as the discovery of individual anomalous instances within a data series, also known as *pointwise anomaly detection*. Anomaly detection can also be classified as group anomaly detection (GAD), which refers to a set of grouped data points whose general collective behavior differs from normal data patterns Chalapathy et al. (2018) and Yu et al. (2015).

4. Methodology

In this section, we present the novelty of our approach for training a DCRNN model, and subsequently using graph-based anomaly detection to discover cyber-attacks in a microservices application.

4.1. Overview

Graph-based anomaly detection has been applied to many different fields including finance, health care and law-enforcement in the past, even network level IT security Akoglu et al. (2015). As far as we know, the authors for Chen et al. (2019) are the only ones who have previously applied this form of anomaly detection to a microservices application.

As mentioned in Section 1, our approach is loosely similar to Chen et al. (2019). The latter carries out their methodology in a two-stage process. In their first stage, they identify clusters of RPCs related to each other in terms of application functionality, which are subsystems of the application as a whole. In their second stage, a DCRNN was trained for each existing RPC subsystem before making predictions and performing anomaly detection. By contrast, we train a single DCRNN model with a more general unified RPC traffic data set to learn the regular behaviour of the entire application. This eliminates the need to identify subsystems and train a model

for every subsystem. Using a single model to detect anomalous traffic with a more unified data set rather than multiple subsystem models as proposed by the RPC clustering approach promotes simplicity and makes the single model more robust. The novelty of our methodology is that we provide a more simplified and generalized method for training a DCRNN to learn the spatial and temporal dynamics of microservices traffic and apply graph-based anomaly detection.

The remainder of Section 4 outlines our anomaly detection approach. First we outline the process of generating synthetic microservice traffic. Then we describe how a microservices application is modelled as a weighted directed graph. We present how the execution of microservices calls is represented as a traffic matrix. We also define the traffic forecasting problem based on this traffic matrix. We then outline how diffusion convolution is used to model existing spatial dependency structures in graphs and how the DCRNN model captures the spatio-temporal dependencies. The DCRNN model is then used to predict future traffic and finally we outline how anomalous traffic is detected by the divergence of the predicted and the actual traffic.

4.2. RPC traffic generation

In the first step of our approach we used an available microservices application. We created synthetic data sets consisting of microservice RPC traffic data by sending HTTP API requests to said application and recording the resulting traces using a distributed tracing tool. These synthetic data sets can be found in Lee and Jacob (2019). The microservices application we ran is part of an open-source benchmark tool called DeathStarBench developed at Cornell University Architecture and group (0000). This benchmark suite is open-source and its individual applications have been used several times in various works, Gan et al. (2020); Hou et al. (2020); Lazarev et al. (2020); Somu et al. (2020) generally for performance management and root cause analysis of microservices.

4.3. Directed graph representation

In microservices traffic, RPCs are initiated between two different services providing collaborative functionality where one service makes a call to the other. We represent calls from one RPC to another RPC as nodes on a directed graph. In other words, a node on a graph represents a source-destination pair of RPCs. A weighted edge between two nodes exists in the graph when the nodes share either a source or destination RPC. This approach promotes scalability, and highlights the architecture of the microservices application and the different inter-relations between the microservices. The directed graph is, in turn, represented as a weighted adjacency matrix.

More formally, we represent the topology of the application services as a weighted directed graph. This graph, known as **G** is represented mathematically as shown in Eq. (1):

$$G = (N, E, A) \tag{1}$$

where **N** is the set of all unique RPC source-destination pair nodes discovered in the traffic, **E** is the set of all edges formed between RPCs when nodes share either a source or destination value, and $\mathbf{A} \in R^{NxN}$ is a weighted adjacency matrix that represents level of adjacency of each node to each other. When a single relation between two source-destination RPC nodes exists, that relation is assigned a weighted value. Each relation between two differing nodes **a** and **b** is assigned the respective weighted values as follows: when **a** and **b** share the same RPC source or destination there is both an edge from **a** to **b** and from **b** to **a** with a weight of 0.5. When one node's source is another node's destination, there is a dependency edge from **a** to **b** (or from **b** to **a**) with a weight of 1.0. The procedure for constructing this adjacency matrix is displayed in Algorithm 1.

Algorithm 1: Build an Adjacency Matrix.

```
Input: An RPC Node Set: N
Output: The adjacency matrix: 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
            V[j, i] ← 0.5;
            V[i, j] ← 0.5;
        end
        if N[i].src == N[j].dst then
            V[j, i] ← 1;
        end
        if N[i].dst == N[j].src then
            V[i, j] ← 1;
        end
    end
end
```

4.4. Traffic matrix representation

As a microservices application executes, a set of attributes for each node is represented as an N x M matrix where N is the number of vertices in the directed graph and M is the number of attributes for each node. For our work, we are concerned with the particular case where a single attribute representing the application traffic is stored. This value is simply the number of times the RPC pair executes. This traffic matrix is obtained from a log of all RPC calls as follows.

We define a series of T' historical time steps. For the simplicity of our experiment, let each time step be of equal duration. We iterate through the logged RPC calls and for each time step, we compute the traffic on each node, that is the number of times the corresponding source-destination RPC call is executed. Given the directed graph **G**, we are returned a series of traffic matrices at every time step from $X_{t-T'+1}$ to X_t, where $\mathbf{X}_t$ denotes the traffic matrix **X** at time step **t**.

4.5. Traffic forecasting

In the field of mathematics, the definition of **traffic forecasting** is to predict future traffic activity given previously learned traffic derived from a network domain Li et al. (2017). Given the directed graph **G** defined in Section 4.3 and the time series of T' traffic matrices from Section 4.4, our goal for the traffic forecasting problem is to define a function that maps T' RPC graph signals at time step **t** to T future time steps as outlined in Eq. (2):

$$[X^{(t-T'+1)}, \ldots, X^{(t)}; G] \xrightarrow{h(.)} [X^{(t+1)}, \ldots, X^{(t+T)}] \tag{2}$$

4.6. Diffusion convolution

GCNNs are designed to learn complex data representations from graphs. One means of defining such representations is to model the spatial dependency structures of directed graphs. This modelling allows us to capture the stochastic features of the traffic Mallick et al. (2020). In our experiment, we use the DCRNN model to train the RPC traffic modelled on **G** using a *diffusion convolution* methodology by relating the traffic flow to a diffusion process

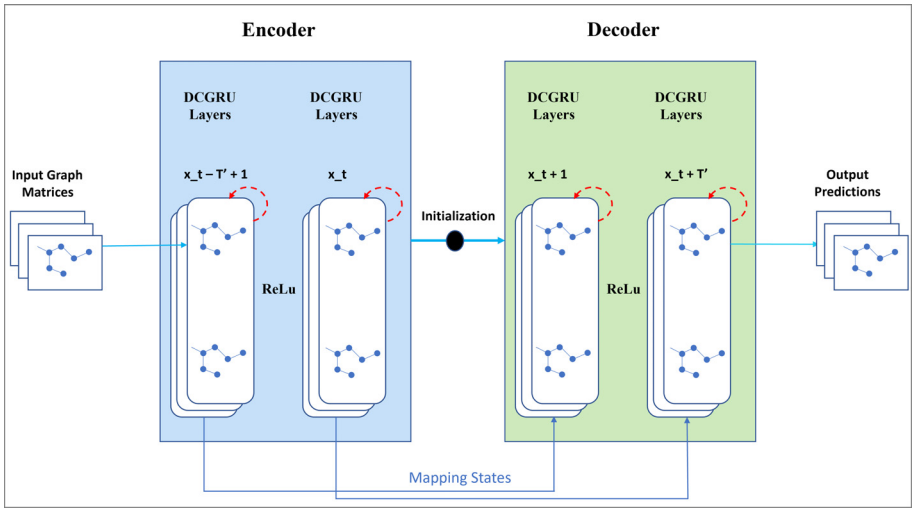

Fig. 1. System Architecture of the Diffusion Convolutional Recurrent Neural Network. The encoder and decoder components are recurrent neural networks composed of DCGRU layers with the *ReLu* activation function. The time series of RPC graph matrices input data is entered into the encoder, trained iteratively using *backpropagation* and the final state is used to initialize the decoder. The decoder then outputs RPC predictions based on ground-truth values at testing time.

Atwood and Towsley (2016). This process can described as a simple random walk on **G** from one node to its neighbor. Furthermore, active traffic flow from a single node to neighbouring nodes can be modelled as a weighted distribution of infinite random walks throughout **G**. We also include the diffusion process in the reverse direction so that model can learn from both upstream and downstream traffic Li et al. (2017). Given **G**, the resulting diffusion convolution operation over traffic attribute matrix **X** is defined as:

$$\mathbf{W}_{\star G}\mathbf{X} = \sum_{d=0}^{K-1}(W_O(D_O^{-1}A)^d + W_I(D_I^{-1}A)^d)X \tag{3}$$

where K represents the maximum number of diffusion steps allowed; **A** represents the adjacency matrix for **G** defined in Section 4.3; $D_O^{-1}A$ and $D_I^{-1}A$ are the transition matrices for the diffusion process and the reverse diffusion respectively; W_O and W_I are the learnable filters applied to the bidirectional diffusion process, and D_I and D_O represent the in-degree and out-degree diagonal matrices which provide the capability to learn from both the upstream and downstream traffic.

4.7. Temporal dependency modelling

To leverage the DCRNN model to capture the temporal dependencies of the microservices traffic, we implement an encoder-decoder architecture of an RNN. The encoder component takes the RPC traffic matrices along the time series as input and the data is encoded into a vector representation. The decoder then reads from this vector and predicts the expected traffic output of future time steps given previously learned ground truth observations.

The Gated Recurrent Unit (GRU) Chung et al. (2014) is a simple, but well-defined variant of an RNN used for designing this encoder-decoder architecture. For our work, both the spatial and temporal dependency modelling are combined by replacing the matrix multiplication functionality of the GRU with the diffusion convolution operation defined in Eq. (3). This leads to the production of the *Diffusion Convolutional Gated Recurrent Unit* (DCGRU). These cells are stacked together to form a series of layers in a sequence-to-sequence fashion to finalize the DCRNN Chan et al. (2020). The architecture of the DCRNN, including the encoder-decoder framework, is displayed in Fig. 1. The functionality defined in Eq. (4) constitutes the DCGRU cell:

$$\begin{aligned}
\mathbf{r}^t &= \sigma(W_{r\star G}[X_t, h_{t-1}] + b_r)\\
\mathbf{u}^t &= \sigma(W_{u\star G}[X_t, h_{t-1}] + b_u)\\
\mathbf{c}^t &= \tanh(W_{c\star G}[X_t(r_t \odot h_{t-1}] + b_c)\\
\mathbf{h}_t &= \mathbf{u}^t \odot h_{t-1} + (1 - \mathbf{u}^t) \odot c_t
\end{aligned} \tag{4}$$

where X_t and h_t represent the input traffic graph matrix and the final state at time step **t** respectively; r^t, u^t and c^t represent the reset gate, update gate and cell state at time step **t**; $\star G$ represents the diffusion convolution operation defined in Eq. (3) and W_r, W_u and W_c are the corresponding filters applied to each equation. This DCGRU cell is used to build the RNN layers. These layers allow the DCRNN model to train with sequential data and capture long-term dependencies Li et al. (2017) and Mallick et al. (2021).

4.8. Training the DCRNN model

By implementing both the spatial and temporal data modelling described above, the *DCRNN* is trained to learn both the spatial dynamics of the adjacency matrix **A** defined in Section 4.3 and the temporal dependencies within the time series from Section 4.4 simultaneously Li et al. (2017).

During the training phase, the adjacency matrix **A** and the time series of traffic matrices are fed into the DCRNN's encoder component as input and the final state at the time step **t** is used to initialize the decoder component as illustrated in Fig. 1. To discover the temporal dependencies and predict the future time series, the RNN layers, composed of DCGRU cells, are trained using backpropagation through time, where the states and input data are trained by the model iteratively over a number of epochs. Finally, the decoder predicts the output for $\boldsymbol{T}$ future time steps. In the testing phase, the ground truth observations are replaced by output predictions generated by the trained model. The DCRNN model is then evaluated by learning the weight matrices in Eq. (3) by minimizing the *mean absolute error* (MAE) loss function:

$$\mathbf{MAE} = \frac{1}{s}\sum_{i=1}^{s}|y_i - \hat{y}_i| \tag{5}$$

where s represents the number of data samples, y_i is the observed ground truth value and $\hat{y}_i$ is the prediction returned by the model for the ith training data sample.

4.9. Anomalous RPC detection

After the DCRNN model returns RPC traffic predictions for testing data, similar to Chen et al. (2019), we perform anomaly detection in order to detect irregular RPC node traffic. The most suitable way to do this is to define a threshold value on the prediction error which is the absolute difference between the ground truth values and the output predictions. Given that X_i^t is the value of RPC node i at time step **t**, the respective prediction error is calculated where $E_i^t = X_i^t - \tilde{X}_i^t$. These thresholds are defined as follows where H_i is an upper and lower threshold for node **i**:

- calculate both the mean $\mu_i = \frac{1}{n}\sum_{t=i}^{n} x_i$ and standard deviation $\sigma_i = \sqrt{\frac{(x_i-\mu_i)^2}{n}}$ for E_t
- set the upper and lower limits for the distribution of each RPC node entry in E_t using the following formula $H_i = \mu_i \pm (2 * \sigma_i)$

5. Experiment

In this section, we outline the microservices application selected for our work, DeathStarBench and its individual microservices, we describe the software environment and tools used for the instrumentation of our application, present the results of our experiment and finally we outline the software libraries and hardware used to carry out the experiment. We present our main work, in which three cyber-attacks are simulated against the application in a series of experiments using penetration testing. For each experiment, the microservices traffic flow was monitored and logged using distributed tracing and the data sets of microservice application activity were generated using the distributed traces. Subsequently, the DCRNN model was trained and evaluated using the data as described in Section 4.8 and the threshold-based anomaly detection methodology defined in Section 4.9 was applied to detect the cyber-attacks. We observe that because each attack is comprised of a large number of concurrent API requests, we can detect a group anomaly by calculating the computed traffic within a specified time window.

5.1. DeathStarBench

The microservices application used for our experiment was DeathStarBench, an open-source benchmark suite comprised of several microservice-based applications. Available end-to-end applications include a social networking app, a banking system service, a media service where a user can post movie reviews and a hotel reservation service Gan et al. (2019a). For this experiment, we selected the social networking application, known as *SocialNetwork*. The DeathStarBench suite itself was written in several different programming languages including Python 3.7, node.js, Java, JavaScript, PHP, C and C++.

The social networking app emulates a broadcast-style network comprised of registered users and the follow relationships from one user to another. This social networking application supports the following actions: a registered user logging in with their credentials, uploading posts embedded with text, hyper-links or other content, broadcasting said posts to other users, a user reading their fellow users' activity and receiving recommendations on what user(s) to follow (Gan et al., 2020). The overall architecture of the application is displayed in Fig. 2.

When the social network application is running, a client user sends a HTTP URL request which is received by a load balancer and web browser component implemented by *Nginx*. The following are API requests that can be sent to the app's logic over the load balancer:

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

The web server delegates these requests to the appropriate microservice to perform requested functionality. Additional services for other operations can also be subsequently called such as database storage, search queries and other functionality. The application's backend server uses **MongoDB** for persistent storage of user profiles, posts, media content and user recommendations, and the data structure stores **Memcached** and **Redis** for caching.

5.2. Software environment & tools

Here, we outline the software environment in which the microservices application was executed for our experiment, especially the software tools used to instrument the application.

5.2.1. Docker

Docker is a set of Platform-as-a-Service (PaaS) tools used for developing, executing and deploying containerized software applications to a virtual software environment (IBM, 2021). Docker is primarily used to combine and store an individual's source code along with their required tools, libraries and settings in standardized executable packages called **containers**. These isolated containers are used to deliver reusable light-weight functionality, which is suitable for implementing the microservices architecture (Jaramillo et al., 2016). In our experiment, each individual microservice for the application is run as a Docker container instance.

5.2.2. Thrift

Thrift is a binary communication protocol developed by the Apache Software Foundation that is used for creating and defining services including user-defined operations and objects (Slee et al., 2007). Thrift functions as an interface definition language which allows the defined services to operate and interface with other services developed across several different programming languages. The Thrift language also provides a scalable framework for client-server RPCs, which is suitable for RPCs sent to microservices (Gan and Delimitrou, 2018). For our experiments, the interfaces that represent each microservice in the application are defined using a *.thrift* written in the Thrift language and the different services communicate with each other using Thrift RPCs.

5.2.3. Jaeger

Jaeger is an open-source software tool used for tracing the execution path of microservices calls propagating throughout a distributed application in response to a user's HTTP request (Authors, 2021). In our experiment, the application is configured to use the Jaeger tool in the form of three separate components each with its own functionality. The **jaeger-agent** is a network daemon that listens for microservice calls, or spans, over a User Data Protocol (UDP) connection; the **jaeger-collector** collects and stores the span data and the **jaeger-query** functions as a query for the jaeger-collector and a UI for accessing and observing the returned traces. We also configured the jaeger-collector to use the multi-tenant **ElasticSearch** API as a storage backend for the returned traces as JSON documents.

5.3. Data generation

Once the social networking application was operationalized, we performed initial work to construct the topology of the social network. We registered 960 users, established 18,800 follow relationships between various users, and constructed a directed graph

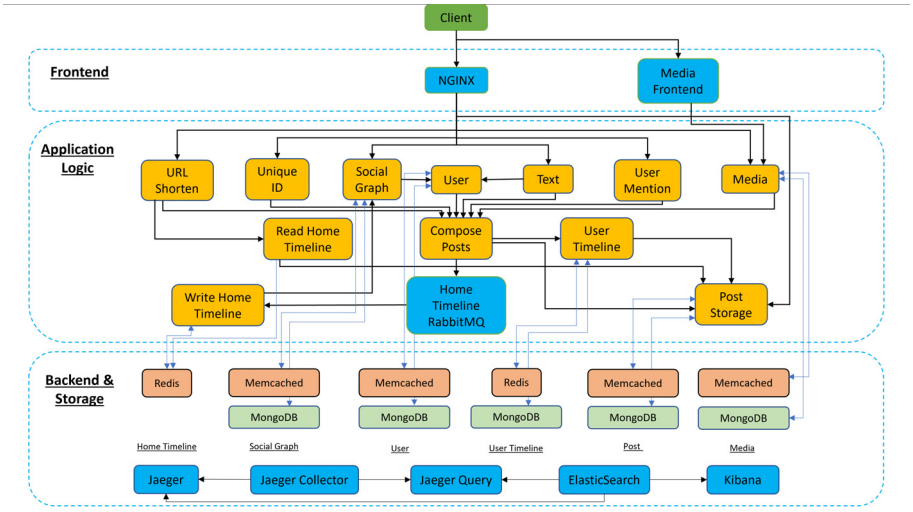

Fig. 2. The Microservices Architecture for the Social Networking Application. On the client side, a user's HTTP request is be forwarded to the application's Nginx web browser. Application requests are then delegated to the various microservice interfaces in the application logic. On the server side, persistent storage of data is handled by **MongoDb** and data caching is handled by **Redis** or **Memcached**. In this application, microservice calls recorded using the **Jaeger** daemon, the data is stored in the **Jaeger Collector** and the **Jaeger Query** sends queries to the collector component. The Jaeger instrumentation also uses the **ElasticSearch** API and the **Kibana** interface for storing and viewing the data as JSON documents respectively.

where the nodes represent the users, and the edges represent the user-follow relationships. Subsequently, we ran microservices calls using API requests and HTTP workload traffic generators to generate both regular application traffic and simulate real world cyber-attacks against the application.

For our work, the social networking application was executed in three separate experiments each with a different cyber-attack. For each experiment, both the regular application traffic and the anomalous traffic caused by the seeded cyber-attacks were generated. The regular application traffic was generally composed of RPC traffic returned in response to API requests to upload users' posts to the application using the API request call *wrk2-api/post/compose*. The RPC nodes for the post composition functionality are listed in Table A.4. Other RPC calls contained in the traffic include calls to register new users, for users to log into the app and read up on fellow users' timelines.

In each experiment, we generated synthetic data sets of timestamped, distributed traces resulting from the microservice RPC traffic sampled over 2 minutes. The resulting data sets contained approximately 18,500 microservice calls. From these data sets, we extracted a total of 63 unique microservice source-destination pair nodes. Section Appendix A Using this set of nodes, we produced a directed graph representation of the application's architecture **G** as outlined in Section 4.3 and from there, an adjacency matrix using Algorithm 1. Subsequently, as outlined in Section 4.4, we extracted a series of time windows from the traffic data and computed the traffic for each RPC node in every designated time window. We set aside 85% of the data as a training set and the remainder was used for testing. To detect the cyber-attacks, we analyzed the RPC traffic within the time window from the testing data set where the cyber-attacks were seeded.

5.4. DCRNN model

For our experiment, the DCRNN model used was composed of 2 recurring DCGRU layers, defined in Section 4.7. These layers were composed of 150 units each and configured with the bidirectional diffusion convolution operation presented in Eq. (3). The maximum diffusion step $\mathbf{K} = 2$, and using the traffic forecasting approach outlined in Section 4.5, the model predicts the RPC traffic matrix

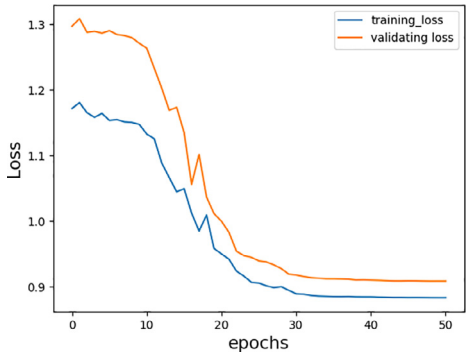

Fig. 3. Learning curves for Training and Validation loss. These loss values were calculated using the **mean absolute error** metric. The loss values were shown to converge nicely over a period of 50 epochs with a divergence smaller than 0.1 in measurement.

for a single future time step. To implement the bidirectional diffusion process defined in Section 4.6, we used the filter type dual-random walk to model the time-series parameters. To load the spatial graph data and build the model, the adjacency matrix defined above was included as a hyper-parameter. For the training process, the hyper-parameters for the DCRNN model that produced the optimal performance are listed as follows: the base learning rate = 0.01, the learning rate decay ratio = 0.1, the *Adam* optimizer was used and the DCRNN model was trained over 50 epochs with an early stopping mechanism set after 15 epochs.

To investigate the effect of spatial and temporal modelling by the DCRNN, we evaluate the model's performance by computing its MAE metric defined in Eq. (5) as a measure of the model's output prediction error. To perform this evaluation, given the training data set defined above, we set aside 70% of the data set to calculate the training loss, and the remainder was used to calculate the validation loss. The learning curves for the metric is displayed in Fig. 3 which shows that the loss values starts out as moderately high, gradually lowering until both curves flatten. Because the gap between the training loss and the validation loss is small, and both

values converge to a point of stability, we observe that the DCRNN model proves to be a reasonably good fit for the data.

5.5. Cyber attacks

We describe the cyber-attacks investigated and executed using penetration testing, and the effect each attack has on the social networking application. To seed these attacks within the microservice RPC traffic in each experiment, scripts were defined to send user HTTP requests to simulate each attack as they would occur in real-life.

5.5.1. Brute force password attack

As we outlined in our earlier work (Jacob et al., 2021), a password guessing, or brute force attack, is an attempt to gain unauthorized access to an online system by systematically guessing passwords until a correct one is found (Dhanabal and Shantharajah, 2015). This results in an abnormally large quantity of incorrect login traffic over a short period of time. This type of cyber-intrusion is detected by monitoring the incorrect login application logs. A viable countermeasure for this cyber-attack is that many applications will have an account lockout policy where a specified number of incorrect login attempts to a single account over a short period of time will result in the account being locked out (Conrad et al., 2016).

The microservice functionality of the application to log into a user's account is called via the *api/user/login* API call which is delegated to the microservice *user-service*. The following lists the steps and sequence of microservice operations called in response to the API *api/user/login*:

- **Login**: the *user-service* calls this operation in response to a API call
- **MmcGetLogin**: the application checks if user's credentials are cached in Memcached
- **MongoFindUser**: user is logging in for the first time and searches for credentials in MongoDB
- **MmcSetLogin**: the application has found user in MongoDb and caches the user in Memcached

If a user logs into the social network application correctly for the first time, the API request sequence ends with the operation **MmcSetLogin** and the application caches their verified login credentials. For a subsequent login, the API sequence ends with the operation **MmcGetLogin** function. An incorrect login request concludes with either the operation **MmcGetLogin** if the user had correctly logged in before or **MongoFindUser** if they were logging in for the first time.

A brute force attack can be used to a crack a password using any possible combination of keyboard characters including letters, numbers, and special characters. This approach results in hundreds of login requests per second (Varonis, 0000). In this experiment, a brute force attack was injected into the testing data set at a single observed time window **t**. This malicious form of computing will enter every existing keyboard character as an attempt at a password and will be composed of ninety-two incorrect login requests. The resulting prediction error for a set of eight randomly selected RPC pair nodes at **t** is displayed in Fig. 4, including the random RPC nodes and their irregular traffic which represent the brute force attack and a number of select nodes from the regular application functionality to compose and upload user's posts.

In Fig. 4, we observe the traffic for the prediction error values for a subset of RPC nodes at time step **t** and the threshold h_i for each node. We see that for a select number of these displayed nodes, their returned traffic exceeds their defined thresholds. These anomalous nodes represent the login functionality

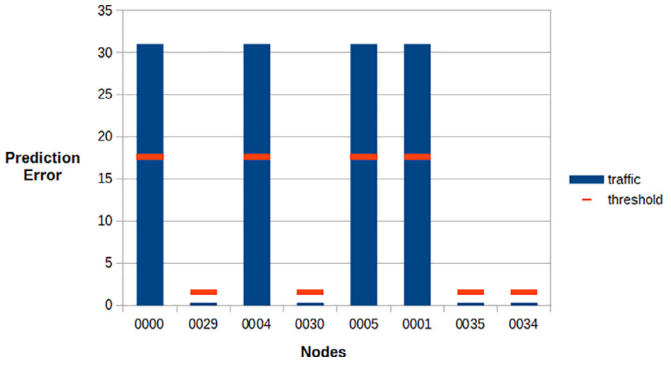

Fig. 4. Brute Force Password Guessing Attack. The application traffic for the RPC nodes that represent the login RPC functionality are shown to exceed their defined thresholds while the RPC traffic for composing a user's posts do not meet their defined threshold.

called during the attack and are referenced in Table A.1 and their respective functionality is outlined as follows:

- **0000**: a client user makes a request to the application with the *api/user/login* API call
- **0001**: the application directs the user's request to the nginx load balancer
- **0004**: the application accepts a request by a user that previously logged in and calls operation **Login**
- **0005**: the app checks if a user's credentials are cached in the Memcached and calls operation **MmcGetLogin**

As indicated above, the RPC traffic prediction error traffic for these four RPC nodes exceed their defined thresholds compared to the random nodes from the regular RPC data listed in Table A.4. Therefore, the conclusion of this experiment was that the injected brute force attack was successfully detected.

5.5.2. Batch registration of bot accounts

A batch registration attack is a form of illegal computing behaviour where a hacker creates multiple fake user accounts, or bot accounts, in large quantities (Chen et al., 2019). These fake accounts are used for a variety of purposes, usually innocuous actions such as the falsely increasing the number of 'likes' on a Facebook page, or more malicious ones like spreading malware on a system application for hacking services and for fraudulent online activity to sway political opinion. Sciences (0000) stated that a means of protecting one's social media application from bot account creation is to establish a baseline of regular application activity and then observe abnormal requests that indicate bot attacks i.e., mass user account creation originating from the same IP address (Pathak, 2014).

In the social networking app, the API request *wrk-api/user/register* creates a new user profile. Therefore, by calling a multitude of these API requests within a short time span, a batch registration attack can be seeded amongst the microservices traffic. The related RPC node IDs that comprise the functionality to register a new account are listed in Table A.2. Like the login API request, the request to register a new user is delegated to the app's *user-service* and the operation **RegisterUserWithId** is called.

Typically, bot accounts are created in tens or hundreds (Pathak, 2014). In this experiment, we seeded a set of *wrk2-api/user-timeline/read* API requests at time step **t** to create 100 new accounts. The resulting prediction error values for a subset of eight random RPC nodes at time step **t** were set aside for evaluation. These nodes were comprised of traffic data resulting from the batch registration attack and regular app functionality and are displayed in Fig. 5 as well as their respective defined thresholds.

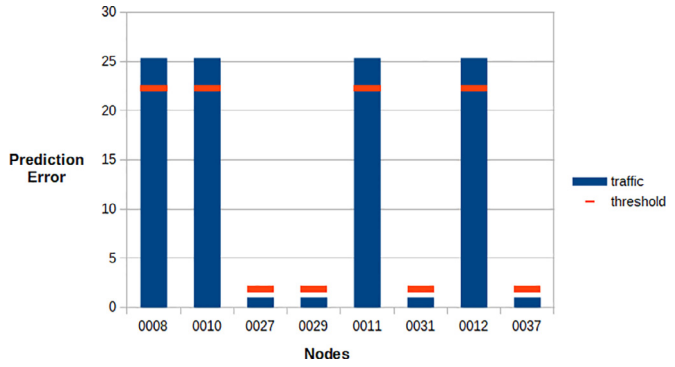

Fig. 5. Batch Registration of Bot Accounts. The traffic values for the RPC nodes that represent the account creation are shown to exceed their thresholds while the nodes for composing a user's posts do not meet their defined threshold.

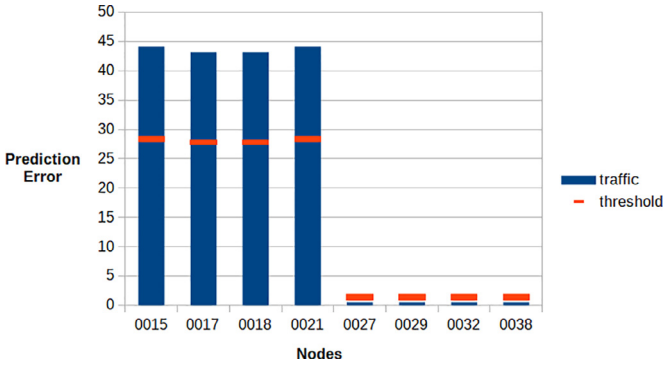

Fig. 6. Distributed denial of service HTTP Flood Attack. The traffic for the RPC nodes to return a users activity timeline exceed their set threshold values while the nodes for composing a user's posts do not meet their threshold values.

As illustrated in Fig. 5, the prediction error for the nodes that represent the batch registration attack exceed their set thresholds, in contrast to the normal microservices data. These anomalous nodes are listed in Table A.2 and outlined as follows:

- **0008**: a client user makes a request to the app with the *wrk2-api/user/register* API call
- **0010**: the nginx load balancer calls the operation **RegisterUser**
- **0011**: app delegates the request to the *user-service* and calls the operation **RegisterUserWithId**
- **0012**: the *user-service* calls an operation **MongoInsertUser** to register the new user with MongoDb

We determine that these anomalous prediction error values for the nodes in the Fig. 5 are the result of the simulated batch registration, showing that the batch registration was detected.

5.5.3. Distributed denial of service

A third cyber-attack explored in our work is a distributed denial of service (DDoS) attack. The attacker aims to overwhelm a system's resources with multiple executions of requests that leaves a service unavailable to legitimate users. A DDoS attack can be categorized as one of three types: volumetric, protocol or application-layer. The application-layer variety supports requests sent to web servers over HTTP and the magnitude of such attacks are measured in requests per second (Revuelto et al., 2017). The DDoS attack we performed in our project was a HTTP Flooding attack which targets both web servers and application-level features. This form of cyber intrusion can be comprised of multiple HTTP **GET** or **POST** requests which can collectively cause a denial-of-service effect (Radware, 2021).

In the social network application, an available GET request, *wrk2-api/user-timeline/read*, can be called to return a timeline of a user's application activity including the composition of uploaded text posts to the application by said user. The web-server delegates this API request to the microservice *user-timeline-service*. The resulting RPC then calls the operation **ReadUserTimeLine** in response to this API request.

The magnitude of application-layer attacks are between 50 and 100 requests per second (Imperva, 2021). For this experiment, we injected a HTTP Flood attack into the microservices RPC traffic composed of 100 GET requests for users' application timelines to simulate a Denial-of-Service intrusion on the application. In a real-life scenario, this attack causes a disruption to the app's *user-timeline-service* microservice and hinder the service operations to store and cache said user timelines in the **MongoDB** and **Redis** services respectively.

After the prediction error was returned by the trained model, values for a number of random RPC nodes that comprise the DDoS attack were compared against random nodes from the regular microservices traffic. These RPC nodes that comprise the DDoS attack are outlined in Table A.3. The prediction error values for these selected nodes and their set thresholds for time step **t** are displayed in Fig. 6.

In Fig. 6, we observe the prediction error for a number of selected nodes resulting from the DDoS attack at time step **t** exceed their node's set thresholds and are described as follows:

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

From these observations, it was determined that the anomalous RPC node traffic to read a user's timeline as part of the simulated HTTP Flooding Attack was detected.

5.6. Software & hardware environment

The microservices application was operationalized using an Intel Core i3-2370 CPU processor. The real-world cyber security attacks were injected into the microservices traffic, and the RPC data was generated also using this device. The actual coding for the experiment was executed using scripts written in Python 3.7+ and included various software libraries including Tensorflow 1.13 and Numpy 0.19.0. The DCRNN model was trained using an NVIDIA GPU server with a four-card Tesla SXM2.

6. Limitations

For our experiment, we train a single DCRNN model to learn the behaviour of the entire microservices application. It is normally necessary to model the entire application as it is difficult to determine where an attacker will target the application.

However, one drawback to our approach is that it is difficult to maintain a large and unified model of general microservice call traffic. Microservices are updated and old functionality is deprecated regularly, so there is a need to retrain the entire model which costs time and resources. As mentioned in Section 1, Chen et al. (2019) uses multiple DCRNN models to learn a different subsystem of the application's functionality which takes less time to train. A limit to this is that their approach would not be able to

detect cyber-attacks which would span multiple subsystems of the application. This is not a limitation of our approach.

In our approach, we monitor the traffic flow of the application to detect cyber-attacks simulated using the microservice HTTP API calls which are sent over the application layer of the system. A cyber-attack that does not take place over the application layer is a Man-in-the-Middle (MITM) attack. A MITM involves an attacker intercepting the traffic between a legitimate user and the application by placing malicious WiFi hotspots on the network layer. Our distributed tracing tool would not be able to monitor the resulting activity caused by the attacker. This cyber-attack can be mitigated by running the application on a Virtual Private Network (VPN) that provides cyber security by encrypting the networking traffic, including the internet connection.

A means of performing a DoS attack against the application is to carry out a TCP-SYN flooding attack which establishes multiple client-server connections. In the case of our application, the attacker creates multiple TCP connections with the application's Nginx web server to deny legitimate users access. Our approach would not be able to detect this attack because it takes place at the transport layer. This attack can be mitigated by implementing SYN cookies, a form of cryptographic hashing that verifies the client as legitimate (Imperva, 0000).

Our ML approach proposes a group-based anomaly detection method that detects attacks that generate a high volume of the same RPC in a small time window. An attack our approach would not be able to detect is a cross-site scripting (XSS) attack. In the DeathStarBench application, a malicious user could upload a post containing a malicious web link that a legitimate user clicks resulting in that user's privileged information being stolen. Our method would not be able to detect this form of intrusion since it requires only a single *composePost* API request. A countermeasure used against this cyber-attack is to use appropriate HTTP response headers to prevent HTTP responses that contain any HTML or JavaScript (PortSwigger, 0000).

Our anomaly detection method is data-driven and relies on machine learning. If the ML-based model or the data are not protected, the security of the model is at risk of becoming a soft spot. Poisoning Apruzzese et al. (2019), as an example, is a type of adversarial machine learning method that perturbates training data in order to poison the model, therefore affecting the decision-making at run-time by altering the threshold value for the anomaly detection model. The corresponding defences can be realized by protecting both data and/or the model. For data, sanitization detects and removes anomalous samples based on predefined rules. In addition, sanitization works on trained models by continuously training the model on new data, thereby mitigating the impact of poisoned samples. For our application, both data and model sanitization are not feasible as it is difficult to define rules and no new training data is available. Another promising countermeasure is to train the model using data from randomized sources which make it difficult for the hacker to devise an effective adversarial attack (Joseph et al., 2013).

In our approach, the anomaly detection threshold value is derived by minimising the prediction error for every node at run-time. A second form of adversarial machine learning attack that violates the model's integrity can be planted at testing time when the trained DCRNN model is operational. The anomaly detector has been deployed, so the attacker aims to subvert its behaviour by modifying malicious ground truth values to not meet the threshold setting and would not be detected as an anomaly. A viable countermeasure is to smooth the decision boundaries of the model, thus reducing the effects caused by adversarial attacks (Xu et al., 2009).

7. Conclusion

This work established that the polylithic behaviour of a microservices-based application can be modelled as a microservice call graph and a distributed tracing tool can be used to monitor users' API requests to the application. We proposed that a state-of-the-art graph convolution network, the Diffusion Convolutional Recurrent Neural Network, can be trained to learn the microservice traffic and discover the spatial and temporal dependencies within the data. Our aim was to perform traffic forecasting to predict future microservice traffic for future time steps. We evaluated the performance of the DCRNN by applying threshold-based anomaly detection to detect abnormal microservices activity that indicated the presence of cyber security attacks. This paper is a continuation of our previous paper where distributed tracing can be used to detect cyber security attacks in microservices.

Using this approach, we detected three different forms of cyber-attack against our application: a brute force attack, a batch registration of bot accounts and a DDoS HTTP Flooding attack. An anomaly detection method was applied by calculating the mean and standard deviation. The microservice traffic resulting from each attack was compared to the normal application traffic for regular application requests at run-time. The difference in RPC traffic volume was proven to be greater than two standard deviations outside the mean which satisfies the empirical rule. Because these attacks can be identified by calculating the greater, irregular volume of the microservice call traffic, our anomaly detection method can be classified as a group anomaly detector.

For this work, the DCRNN model was used to carry out RPC traffic prediction. The model predicted the number of times an RPC is called in a specific time period. Distributed tracing can also be used to capture the duration of the microservice calls. Therefore, future work could include the detection of irregular microservice activity based on their execution time.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Stephen Jacob: Writing – original draft, Methodology, Software, Data curation, Formal analysis, Visualization, Investigation, Resources. **Yuansong Qiao:** Validation, Software, Resources. **Yuhang Ye:** Writing – review & editing, Validation. **Brian Lee:** Conceptualization, Writing – review & editing.

Acknowledgments

This publication has emanated from research conducted with the financial support of Athlone Institute of Technology under its Presidents Seed Fund (2021) and Science Foundation Ireland (SFI) under Grant Number SFI 16/RC/3918, co-funded by the European Regional Development Fund.

Appendix A

Table A1
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 A2
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 A3
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 A4
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

(*continued on next page*)

Table A4 (*continued*)

ID	Source	Destination
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 + UploadUserMentions	post-storage-service + StorePost
0046	compose-post-service + UploadUserMentions	user-timeline-service + WriteUserTimeline
0047	compose-post-service + UploadUserMentions	write-home-timeline-service + FanoutHomeTimelines
0048	unique-id-service + UploadUniqueId	compose-post-service + UploadUniqueId
0049	compose-post-service + UploadUniqueId	compose-post-service + RedisHashSet
0050	compose-post-service + UploadText	compose-post-service + RedisHashSet
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 + WriteUserTimeline
0054	write-home-timeline-service + FanoutHomeTimelines	social-graph-service + GetFollowers
0055	write-home-timeline-service + FanoutHomeTimelines	write-home-timeline-service + RedisUpdate
0056	post-storage-service + StorePost	post-storage-service + MongoInsertPost
0057	social-graph-service + GetFollowers	social-graph-service + RedisGet
0058	social-graph-service + GetFollowers	social-graph-service + MongoFindUser
0059	social-graph-service + GetFollowers	social-graph-service + RedisInsert
0060	user-timeline-service + WriteUserTimeline	user-timeline-service + MongoFindUser
0061	user-timeline-service + WriteUserTimeline	user-timeline-service + MongoInsert
0062	user-timeline-service + WriteUserTimeline	user-timeline-service + RedisUpdate

References

Akoglu, L., Tong, H., Koutra, D., 2015. Graph based anomaly detection and description: a survey. Data Min. Knowl. Discov. 29 (3), 626–688.

Anodot. What is anomaly detection?(Accessed on 10/27/.2021), https://www.anodot.com/blog/what-is-anomaly-detection/2020.

Apruzzese, G., Colajanni, M., Ferretti, L., Marchetti, M., 2019. Addressing adversarial attacks against security systems based on machine learning. In: 2019 11th International Conference on Cyber Conflict (CyCon), volume 900. IEEE, pp. 1–18.

Architecture S., group) I.L.S.. Github - delimitrou/deathstarbench: Open-source benchmark suite for cloud microservices. https://github.com/delimitrou/DeathStarBench, (Accessed on 01/27/2022).

Atwood, J., Towsley, D., 2016. Diffusion-convolutional neural networks. In: Advances in neural information processing systems, pp. 1993–2001.

Authors T.J.. Jaeger: open source, end-to-end distributed tracing. (Accessed on 10/27/2021), https://www.jaegertracing.io/; 2021.

Chalapathy, R., Toth, E., Chawla, S., 2018. Group anomaly detection using deep generative models. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, pp. 173–189.

Chan, V., Gan, Q., Bayen, A., 2020. A graph convolutional network with signal phasing information for arterial traffic prediction. arXiv preprint arXiv:201213479.

Chen, J., Huang, H., Chen, H., 2019. Informer: irregular traffic detection for containerized microservices RPC in the real world. In: Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, pp. 389–394.

Chung, J., Gulcehre, C., Cho, K., Bengio, Y., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:14123555.

Conrad, E., Misenar, S., Feldman, J., 2016. Eleventh Hour CISSP®: Study Guide. Syngress.

Dhanabal, L., Shantharajah, S.P., 2015. 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.

Gan, Y., Delimitrou, C., 2018. The architectural implications of cloud microservices. IEEE Comput. Archit. Lett. 17 (2), 155–158.

Gan, Y., Zhang, Y., Cheng, D., Shetty, A., Rathi, P., Katarki, N., Bruno, A., Hu, J., Ritchken, B., Jackson, B., et al., 2019. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 3–18.

Gan, Y., Zhang, Y., Cheng, D., Shetty, A., Rathi, P., Katarki, N., Bruno, A., Hu, J., Ritchken, B., Jackson, B., et al., 2020. Unveiling the hardware and software implications of microservices in cloud and edge systems. IEEE Micro 40 (3), 10–19.

Gan, Y., Zhang, Y., Hu, K., Cheng, D., He, Y., Pancholi, M., Delimitrou, C., 2019b. Leveraging deep learning to improve performance predictability in cloud microservices with seer. ACM SIGOPS Oper. Syst. Rev. 53 (1), 34–39.

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Comput. 9 (8), 1735–1780.

Hou, X., Li, C., Liu, J., Zhang, L., Hu, Y., Guo, M., 2020. Ant-man: towards agile power management in the microservice era. In: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, pp. 1–14.

IBM I.. What is docker?(Accessed on 10/27/2021),https://www.ibm.com/in-en/cloud/learn/docker; 2021.

Imperva. (1) new messages!https://www.imperva.com/learn/ddos/syn-flood/ (Accessed on 01/31/2022).

Jacob, S., Qiao, Y., Lee, B.A., 2021. Detecting cyber security attacks against a microservices application using distributed tracing. In: ICISSP, pp. 588–595.

Jaramillo, D., Nguyen, D.V., Smart, R., 2016. Leveraging microservices architecture by using docker technology. In: SoutheastCon 2016. IEEE, pp. 1–5.

Joseph, A.D., Laskov, P., Roli, F., Tygar, J.D., Nelson, B., 2013. Machine learning methods for computer security (dagstuhl perspectives workshop 12371). In: Dagstuhl Manifestos, volume 3. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, pp. 1–30.

Imperva. What does DDoS mean? | distributed denial of service explained | imperva. (Accessed on 09/30/2021), https://www.imperva.com/learn/ddos/denial-of-service/?utm_campaign=Incapsula-moved; 2021.

Kung-Hsiang H.T.D.S.. A gentle introduction to graph neural networks (basics, deepwalk, and graphsage). (Accessed on 09/17/2021), https://towardsdatascience.com/a-gentle-introduction-to-graph-neural-network-basics-deepwalk-and-graphsage-db5d540d50b3; 2019.

Lazarev, N., Adit, N., Xiang, S., Zhang, Z., Delimitrou, C., 2020. Dagger: towards efficient rpcs in cloud microservices with near-memory reconfigurable nics. IEEE Comput. Archit. Lett. 19 (2), 134–138.

Le, D.Q., Jeong, T., Roman, H.E., Hong, J.W.K., 2011. Traffic dispersion graph based anomaly detection. In: Proceedings of the Second Symposium on Information and Communication Technology, pp. 36–41.

Lee B., Jacob S.. [dataset] | gitlab | stephenj - repository. (Accessed on 02/04/2022),https://gitlab.com/sri-ait-ie/phd-projects/stephenj/-/tree/journal_Branch; 2019.

Lee, J., Bae, H., Yoon, S., 2020. Anomaly detection by learning dynamics from a graph. IEEE Access 8, 64356–64365.

Li, Y., Yu, R., Shahabi, C., Liu, Y., 2017. Diffusion convolutional recurrent neural network: data-driven traffic forecasting. arXiv preprint arXiv:170701926.

Lv, Y., Duan, Y., Kang, W., Li, Z., Wang, F.Y., 2014. Traffic flow prediction with big data: a deep learning approach. IEEE Trans. Intell. Transp. Syst. 16 (2), 865–873.

Ma, X., Dai, Z., He, Z., Ma, J., Wang, Y., Wang, Y., 2017. Learning traffic as images: a deep convolutional neural network for large-scale transportation network speed prediction. Sensors 17 (4), 818.

Mallick, T., Balaprakash, P., Rask, E., Macfarlane, J., 2020. Graph-partitioning-based diffusion convolutional recurrent neural network for large-scale traffic forecasting. Transp. Res. Rec. 2674 (9), 473–488.

Mallick, T., Balaprakash, P., Rask, E., Macfarlane, J., 2021. Transfer learning with graph neural networks for short-term highway traffic forecasting. In: 2020 25th International Conference on Pattern Recognition (ICPR). IEEE, pp. 10367–10374.

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

Polato, M., Sperduti, A., Burattin, A., de Leoni, M., 2018. Time and activity sequence prediction of business process instances. Computing 100 (9), 1005–1031.

PortSwigger. What is cross-site scripting (XSS) and how to prevent it? | web security academy. https://portswigger.net/web-security/cross-site-scripting, (Accessed on 01/31/2022).

Radware. Http flood (http ddos attack). (Accessed on 09/13/2021), https://www.radware.com/security/ddos-knowledge-center/ddospedia/http-flood/; 2021.

Revuelto S., Socha K., Meintanis S., 2017. DDoS overview and response guide. https://cert.europa.eu/static/WhitePapers/CERT-EU_Security_Whitepaper_DDoS_17-003.pdf, (Accessed on 09/13/2021).
Sciences S.. What are bot attacks? Bot mitigation for web apps & APIs. https://www.signalsciences.com/glossary/bot-attack-protection/, (Accessed on 09/16/2021).
Slee, M., Agarwal, A., Kwiatkowski, M., 2007. Thrift: scalable cross-language services implementation. Facebook white paper 5 (8), 127.
Somu, N., Daw, N., Bellur, U., Kulkarni, P., 2020. Panopticon: A comprehensive benchmarking tool for serverless applications. In: 2020 International Conference on COMmunication Systems & NETworkS (COMSNETS). IEEE, pp. 144–151.
Sun, Y., Nanda, S., Jaeger, T., 2015. Security-as-a-service for microservices-based cloud applications. In: 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, pp. 50–57.
Tax, N., Verenich, I., La Rosa, M., Dumas, M., 2017. Predictive business process monitoring with LSTM neural networks. In: International Conference on Advanced Information Systems Engineering. Springer, pp. 477–492.
Varonis. What is a brute force attack?https://www.varonis.com/blog/brute-force-attack, (Accessed on 01/21/2022).
Wu, Y., Tan, H., 2016. Short-term traffic flow forecasting with spatial-temporal correlation in a hybrid deep learning framework. arXiv preprint arXiv:161201022.
Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y., 2020. A comprehensive survey on graph neural networks. IEEE Trans. Neural Netw. Learn. Syst. 32 (1), 4–24.
Xu, H., Caramanis, C., Mannor, S., 2009. Robustness and regularization of support vector machines. J. Mach. Learn. Res. 10 (7).
Yao, Y., Su, L., Lu, Z., Liu, B., 2019. Stdeepgraph: Spatial-temporal deep learning on communication graphs for long-term network attack detection. In: 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE). IEEE, pp. 120–127.
Yu, B., Yin, H., Zhu, Z., 2017. Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. arXiv preprint arXiv:170904875.
Yu, R., He, X., Liu, Y., 2015. Glad: group anomaly detection in social media analysis. ACM Trans. Knowl. Discov. Data (TKDD) 10 (2), 1–22.

Stephen Jacob is a Ph.D. candidate with the Software Research Institute (SRI) at the Technological University of the Shannon: Midlands Midwest working in the field of Cyber Security. He received his BsC in Computer Science at University of Limerick in 2015 and his MSc in Software Engineering at Athlone Institute of Technology in 2016.

Dr. Yuansong Qiao is a Senior Research Fellow in the Software Research Institute (SRI) at Technological University of the Shannon: Midlands Midwest, Ireland. He is a Science Foundation Ireland (SFI) Funded Investigator in the SFI CONFIRM Smart Manufacturing Centre. He received his Ph.D. in Computer Applied Technology from the Institute of Software, Chinese Academy of Sciences, Beijing, China, in 2008. He is a member of IEEE (Communications, Computer and Robotics and Automation societies and Blockchain Community) and ACM (SIGCOMM and SIGMM). His research interests include Future Internet Architecture, Blockchain Systems, Robotics and Edge Intelligence and Computing.

Dr. Brian Lee is the director of the Software Research Institute (SRI) at Technological University of the Shannon: Midlands Midwest, Ireland. He is a Science Foundation Ireland (SFI) Funded Investigator in the SFI CONFIRM Smart Manufacturing Centre. He received his Ph.D. in Computer Science from Trinity College Dublin in 2004. He is a member of IEEE (Communications, Computer and Robotics and Automation societies) and ACM). His research interests include Computer Security (Access Control, Network Security, Security Analytics) and Programmable Networking and Edge Computing.

Yuhang Ye received the B.Eng. and M.EngSc. degrees in Electronic Engineering from the National University of Ireland, Maynooth, Ireland, and the PhD degree from the Athlone Institute of Technology, Ireland. He is currently a Post-Doctoral Researcher with the Software Research Institute, Technological University of the Shannon. His current research interests include IIoT security, adversarial machine learning and multimedia communication.