GRU-based Deep Learning Approach for Network Intrusion Alert Prediction

Mohammad Samar Ansari[a], Vaclav Bartos[b], Brian Lee[c]

[a]*Aligarh Muslim University, India*
[b]*CESNET, Czech Republic*
[c]*Software Research Institute, Athlone Institute of Technology, Ireland*

Abstract

The exponential growth in the number of cyber attacks in the recent past has necessitated active research on network intrusion detection, prediction and mitigation systems. While there are numerous solutions available for intrusion detection, the prediction of future network intrusions still remains an open research problem. Existing approaches employ statistical and/or shallow machine learning methods for the task, and therefore suffer from the need for feature selection and engineering. This paper presents a deep learning based approach for prediction of network intrusion alerts. A Gated Recurrent Unit (GRU) based deep learning model is proposed which is shown to be capable of learning dependencies in security alert sequences, and to output likely future alerts given a past history of alerts from an attacking source. The performance of the model is evaluated on intrusion alert sequences obtained from the *Warden* alert sharing platform.

Keywords: Alert Prediction, Cybersecurity, Deep Learning, Network Intrusion Prediction

1. Introduction

In light of the ever increasing network attacks (both in number as well as intensity), it is necessary to continuously monitor and analyze the enormous volumes of data transmitted around the globe. This has led to the development of a variety of network monitoring and analysis systems, including intrusion detection systems (IDS), honeypots, network flow monitors, etc. Such systems are widely deployed in today networks where they help administrators to deal with various attacks and intrusions. Sometimes, data from these systems are also shared among multiple organizations using various data-sharing platforms [1, 2, 3], which allows for more proactive solutions (e.g. blocking the most dangerous attackers seen in other networks) rather reactive ones. In recent years, the availability of large amounts of information about detected cyber attacks, as well as recent advances in machine learning, allowed researches to focus not only on *detection*, but also on *prediction* of cyber attacks [4, 5, 6].

Several predictive methods in the area of cybersecurity has been proposed in recent years [7, 8, 9, 10], and while they are good proofs of concept showing that predicting future attacks is possible, they still have very limited capabilities and, therefore, limited use in practice. For example, they only allow to predict the expected number of detected attacks in a future time interval[7], the most probable next step of an already ongoing multi-stage attack [8], or just the probability there will be some attack originating from a given source [9].

In this paper we propose a novel deep-learning based method for prediction of network attacks coming from a known malicious source, which is capable to predict not only the probability of an attack observation, but rather predicts concrete parameters (e.g. its type, intensity and target) of the expected attack, which enables better defense measures to be applied.

1.1. Motivation

Most of the previous works focus on prediction of future attacks (or rather future steps of a complex attack) against a single *target*. However, as works on *predictive blacklisting* [11, 12] as well as a recent work by Bartos et al. [9] showed, it is also useful to predict future behavior of previously identified malicious *sources*. Such a view can be especially useful in connection to various alert sharing platforms, which are being increasingly used in the last years [1, 13, 14], since it allows to leverage information about attacks against different targets to predict future ones. This is also backed by our own experience as an operator of a nation-level network and a multi-organization alert sharing community (more in Sec. 3). Our task here is not to protect a single network (which is a common task of most other practitioners), but to gather information on attacks and attackers observed in multiple networks, analyze them, and warn end-networks about imminent threats. Prediction of future actions of an attacker is one of the most important goals here.

For example, it is often not possible to blacklist all IP addresses previously reported as a source of some attack, either for technical reasons (e.g. a limited number of firewall rules) or due to a high risk of false positives. Information about the expected behavior of each potentially malicious source, like the probability of further attacks, their type, severity or target, can help to set up better blacklists or other defensive measures (e.g. rate limiting, limit on number of login attempts). A score derived from predicted future behavior of individual IP addresses can also be used to filter traffic during DDoS attacks [15]. Furthermore, information about the predicted continuation or repetition of detected attacks can be very helpful in alert prioriti-

zation algorithms which help human operators to decide which alerts to solve first.

A model presented in [9] only predicts the *probability* of observing future attacks coming from a malicious source in a certain time interval. It is done by using shallow learning and manually engineered features. In this work, we use the more advanced, deep-learning approach – neural network with Gated Recurrent Units (GRU) [16], which is a model able to effectively learn long term dependencies in sequence data. Unlike shallow learners, the learning process for deep learners does not depend on human-crafted (derived) features [17]. This can be attributed to the fact that deep learning has the potential to extract better representations of high-level data (basic features) which is made possible by the inherent complex architecture of the network as well as the possibility of inclusion of non-linear transformations [18, 19]. As a result, our GRU-based model is able to take a sequence of previous alerts with just a little preprocessing and predict concrete properties of the next few alerts, like their category, volume, target IPs or approximate time. This allows to apply defensive measures very precisely.

1.2. Contribution

The contributions of the paper can be listed as follows:

- Prediction of alerts from malicious sources, as opposed to the alert prediction against a given target usually performed in the available literature.
- Prediction of both categorical (e.g. protocol, attack category) and non-categorical (e.g. time of attack, volume) fields in the alert, while most previous works either predict category of future event or probability of a certain event occurring in the future.

This paper is organized as follows. Section 2 discusses pertinent works available in the technical literature. Section 3 contains a description of the data used in this work. Section 4 presents the objectives and goal definition. Section 5 contains the details of data preparation (pre-processing) for making it amenable for the deep learning model. Section 6 presents the details of the proposed deep learning network for alert prediction, with a subsection dealing with the DL model parameter selection and tuning. Section 7 describes the error metric we used to evaluate the prediction model. Results of the evaluation are presented in Section 8. Section 9 highlights some comparisons between the proposed solution and similar existing works. Section 10 mentions the limitations and avenues for future improvements. Section 11 contains concluding remarks.

2. Existing Works

Predictive methods in cyber security can be divided into three main areas by their use case [5]: (i) attack projection and intention recognition, (ii) intrusion prediction and (iii) network security situation forecasting. The task of the methods in the first area is to predict what is an attacker (in an already observed attack) going to do next and what is its ultimate goal. A survey of earlier attack projection methods can be found in [20]. Later examples include [10, 21, 22], which use data-mining methods to infer typical patterns in sequences of IDS alerts to predict next ones; [23, 24, 25] which use Hidden Markov Models to model the propagation of multi-step attacks, or [26] which explores the possibilities of using LSTM networks for the task.

The task of situation forecasting, is to predict some aspects of the overall security situation in the network, such as estimation of the number of observed attacks or vulnerabilities. Examples in this group are [27, 7], both using some sort of time-series analysis, or [28] which uses a complex method to assess the current network security situation and an optimized neural network to predict its next stage.

Our current work belongs to the second of the categories defined above – intrusion prediction. Such methods try to predict what type of attack will occur, when, and where. Works in this area are summarized in surveys [29] and [4]. According to [29], predictive methodologies can be based on alert correlation, sequences of actions, statistical and probabilistic methods, and feature extraction. The prediction systems can be based on hidden Markov models, Bayesian networks, genetic algorithms, data mining or machine learning. Some recent approaches also include non-technical data sources, like sentiment analysis on social networks [30, 31] to predict potential attacks against an organization or changes in user behavior [32], or detection of changes in user behavior to predict insider threats [33].

Regarding previous uses of machine learning methods for attack prediction, most proposed approaches are simple shallow learning methods. To the best of our knowledge, the only works using some kind of deep learning methods, are [8] and [26]. However, both focus mostly on projection of an ongoing attack, rather than prediction of new attacks. A detailed discussion on the shallow and deep learning methods follows.

Machine Learning approaches for intrusion detection include solutions based on K-Nearest Neighbor (KNN) [34], Support Vector Machines (SVM) [35], Naive Bayes (NB) networks [36], Multi-Layer Perceptron (MLP) [37], and Decision Trees (DT) [38]. Amongst the DL-based approaches, Javaid et. al. presented a deep learning based NIDS which used Self-Taught Learning (STL), and evaluated their network on the NSL-KDD dataset [39]. Kim et. al. employed an LSTM-based recurrent neural network architecture for implementing an IDS, and tested it on the KDD Cup 1999 dataset [40]. Roy et. al. explored the use of SVMs and DL architectures for use as a classifier for different types of network intrusion attacks [41]. Yin et.al. proposed a deep learning approach for intrusion detection using recurrent neural networks, and studied the performance of the model in binary multi-class classification on the benchmark NSL-KDD dataset. The authors compared their results with those obtained using random forest, support vector machine, and other existing ML methods, and demonstrated that their network was very suitable as a classification model with higher accuracy and performance in both binary and multi-class classification [19]. Another recurrent neural network (RNN) based alert detection system, referred to as *DeepLog* was put forward by Du et. al. which models a system log as a natural language sequence and learn patterns from normal executions and detect anomalies when log patterns deviate from the model [42].

Kim et. al. used a Deep Neural Network (DNN) with 4 hidden layers and 100 hidden units to form their intrusion detection model, and reported results on the KDD Cup 99 dataset [43]. Shone et. al. recently proposed a non-symmetric deep auto-encoder (NDAE) for unsupervised feature learning, and used multiple NDAEs to construct a DL classification model. The approach was evaluated using the benchmark KDD Cup 99 and NSL-KDD datasets with promising results [44]. Aljawarneh et. al. presented a hybrid model consisting of different classifiers like J48, Meta Pagging, RandomTree, REPTree, AdaBoostM1, DecisionStump and NaiveBayes. Thereafter, the best classifier was chosen by using a voting algorithm with Information Gain that combines the probability distributions of the used learners. The NSL-KDD dataset was used for the evaluation of the method [45].

There have been relatively fewer research attempts to leverage alert data for the prediction of future behavior of malicious source(s). One early and pertinent work in that direction is a series of papers on *predictive blacklisting*, a technique in which data from a large alert sharing platform (DShield) is used to generate 'personalized' blacklists for each contributor, listing those sources that are the most likely to attack the contributor in a given time duration in the future [46, 47]. The works however only focus on building a blacklist, not to characterize the sources in any other way, such as assigning a score or estimating the probability of different types of future alerts. More recent efforts towards prediction of malicious network activities include [48, 49]. Ramaki et. al. proposed an alert correlation framework which correlates the alerts, and constructs attack scenarios using Bayesian networks. Attack prediction rules are formed and subsequently used to forecast the next goal of an attacker [48]. However, prediction results are demonstrated only for the first alert set of DARPA 2000 dataset called the LLDDoS1.0 comprising of only 1813 raw alerts and 1236 meta-alerts. Such a small alert set cannot be considered an actual replication of the real-world scenario. Okutan et. al. presented a cyber attack forecast system called Cyber Attack Prediction of Threats from Unconventional Resources (CAPTURE), which used the Auto-Regressive Integrated Moving Average (ARIMA) model to generate inferences from the alert data, and train a Bayesian classifier. The authors showed that it is indeed possible to forecast future cyber incidents using CAPTURE [49]. More recently, Holgado et. al. proposed a method based on the Hidden Markov Model (HMM) to predict multi-step attacks using IDS alerts, and validated their method using a virtual Distributed Denial of Service (DDoS) scenario [50]. Their approach involves the tuning of the HMM parameters using unsupervised training algorithms *viz.* Baum-Welch, Expectation Maximisation (EM), Generalised EM, and the Gradient Descent method. The approach is applied to DDoS attack scenarios only and results are included for predictions on the LLDDoS1.0 alert set taken from the DARPA 2000 dataset. Wang et. al. proposed a method of cyber-attack prediction based on threat intelligence (TI) wherein a matching method is used to extract high-quality TI from the external alert data, and then predicting the attack behavior based on the context data [51]. The works in [27, 7] use variants of time-series analysis, and the work in [28] employs a complex method to assess the current network security situation and an optimized neural network to predict its next stage.

Closely related is a recent work by Bartos et. al. in which alert predictions are done over the same type of data as in the presented work [9]. However, their method is only focused on estimating the probability of observing a future attack from a given source (using shallow neural networks and gradient boosted decision trees). In the presented work, we go further and try to predict concrete parameters of such an attack, like its type, volume, approximate time and target, for which we use the deep learning approach.

The only existing relevant example of deep learning application for alert prediction is a recent work from Shen et. al., in which authors presented a RNN based solution for predicting future events on a machine based on previous observations [8]. However, that solution was tailored to specialized data collected from machines running Symantec's intrusion detection software. The present work however considers generic fields like time of detection of attack, volume of attack, target IP, port, etc., and therefore is suitable for any alert dataset containing such fields.

3. Alert Data

Alert data used in the paper is obtained from the threat sharing platform *Warden*[1] (also known as SABU platform[2]) – an alert sharing tool and community run by CESNET, Czech National Research and Education Network (NERN) [52]. It is an open-source platform designed for automatic sharing of detected security events amongst Cyber Security Incident Response Teams (CSIRT). Every month, Warden receives (and redistributes) millions of alerts from various IDS, honeypots, network probes, and other detection systems deployed in several university networks, the NREN backbone and also one large commercial provider.

Warden alerts are available in the IDEA format[3] – a simple JSON-based format for description of various security events, which is designed to be easy for humans to read and comprehend, and straightforward for computers to parse and generate. A typical Warden alert contains several fields such as DetectTime, Source IP, Target IP, Port, Protocol, CreateTime, CeaseTime, Category, Detector (Node) Name, Detector (Node) Type, and many others. However, not all fields are present for all alerts with some detectors reporting more/different fields than others. A judicious selection of fields is first required to filter the alert data coming from different detectors to make the pruned dataset consistent across multiple detectors. For the purpose of this work, the alerts were trimmed to have the fields given by the following tuple:

```
{DetectTime, FlowCount, SourceIP(/24),
TargetIP(/24), Port, Protocol, Category, Node,
                Node Type}
```

[1] https://warden.cesnet.cz/

[2] https://sabu.cesnet.cz/en/start

[3] https://idea.cesnet.cz/en/definition

Table 1: A sample of typical Warden alerts. Source IPs and Target IPs are shown after trimming to their respective /24 prefixes. Node names and some Target IPs are partially anonymized here to not reveal sensitive information

DetectTime	**FlowCount**	**SourceIP**	**TargetIP**	**Port**	**Protocol**	**Category**	**Node**	**Node Type**
2019-06-15T21:55:25Z	10	106.12.128		22	tcp	Attempt.Login	xx.yy.nemea.bruteforce	Relay
2019-06-13T21:34:28Z	93	82.204.182		3389	tcp	Attempt.Login	xx.yy.nemea.bruteforce	Relay
2019-06-13T19:13:19Z	26	212.22.78	aa.bb.254	80	tcp	Recon.Scanning	xx.yy.tarpit	Relay
2019-06-12T13:14:01Z	4	187.32.27	aa.bb.252	445	tcp	Recon.Scanning	xx.yy.tarpit	Relay
2019-06-10T18:47:32Z	300	195.113.142	141.74.33		tcp	Recon.Scanning	xx.yy.nemea.vportscan	Relay
2019-06-09T16:05:19+00:00	1					Other Category	xx.yy.supplier.intelmq	Relay
2019-06-08T18:07:35Z	14	205.185.114		22	tcp	Attempt.Login	xx.yy.nemea.bruteforce	Relay
2019-06-08T00:46:55Z	742	195.154.63	160.217.95	5060	udp	Attempt.Login	xx.yy.nemea.sipbruteforce	Relay
2019-06-05T00:06:22+00:00	1	199.189.248				Malware	xx.yy.supplier.intelmq	Relay
2019-06-04T22:11:21Z	27	198.54.117				Intrusion.Botnet	xx.yy.nemea.blacklist	Relay
2019-06-04T17:49:41Z	1	119.28.58	147.230.198	123	udp	Availability.DoS	xx.yy.ids_collector.suricata	Datagram
2019-06-04T09:36:45Z	1	152.89.163	195.113.150	9400	udp	Attempt.Exploit	xx.yy.ids_collector.tippingpoint	Datagram
2019-06-01T22:45:59Z	10	73.243.42		22	tcp	Attempt.Login	xx.yy.nemea.bruteforce	Relay
2019-06-01T00:37:52Z	410	81.215.231				Recon.Scanning	xx.yy.hoststats	Flow

Table 2: Statistics from the Warden alert data for May–August 2019

Parameter	Value				
	May	June	July	August	Total
No. of alerts	61,882,145	56,552,131	65,395,937	78,493,752	262,323,965
No. of SourceIP with ≥40 alerts per month	127,116	117,590	118,883	130,051	493,640
Alerts for all SourceIP with ≥40 alerts per month	58,014,455	52,857,722	61,697,634	74,909,820	247,479,631
Unique SourceIP with ≥40 alerts in 4 months					271,989
Total no. of training and testing vectors generated	56,616,179	51,564,232	60,389,921	73,479,259	242,049,591

where

- `DetectTime`: time of generation of alert, numerical value.
- `FlowCount`: measure of attack intensity such as the number of connection attempts, numerical value.
- `SourceIP(/24)`: IP /24-prefix of the attacking source, hereafter referred to as SourceIP, numerical value.
- `TargetIP(/24)`: IP /24-prefix of the attacked target, hereafter referred to as TargetIP, numerical value.
- `Port`: port accessed on the target (e.g. Port 80, Port 5600, etc.), categorical value.
- `Protocol`: protocol used for carrying out the attack (e.g. TCP, UDP, etc.), categorical value.
- `Category`: attack category (e.g. Reconnaissance, Scan, Attempt Exploit, etc.), categorical value.
- `Node`: name of the detector issuing the alert, categorical value.
- `Node Type`: type of the detector which issued the alert (e.g. honeypot, IDS, flow monitor, etc.), categorical value.

The reasons for identifying the above items for the alert tuple are twofold: firstly, these are the fields which are present in most of the alerts irrespective of the detector, and secondly these fields were identified as more significant than other alert fields such as `ID` (which is an alphanumeric identification tag for the alert), `Format` (which is 'IDEA0' for all the Warden alerts and therefore redundant), `Description` (which essentially is a text description of the contents of the other alert fields), etc. for the characterization of a network intrusion alert. Only the /24 prefixes instead of the complete IP addresses for Target IPs (the IPs of the computer systems, attacks on which which have been detected by the Warden system) are used in the alert tuple because of the fact that attackers are more likely to target a complete subnet instead of a particular IP address. Similarly, only the /24 prefixes for the Source IPs (the IPs of the computer systems carrying out the attacks, which have been detected by the Warden system) were considered to ensure that attacks originating from a particular subnetwork are clubbed together. This was done in the light of the study conducted by Moura et. al. [53], which showed that malicious IP addresses tend to cluster in *bad neighborhoods*, subnetworks with higher proportion of malicious sources than normal. It is also in line with works on *predictive blacklisting* [46, 47], where prediction is only done for /24 prefixes as well . Table 1 presents a sample of the alert data with entries corresponding to the alert tuple chosen above, and considering only the /24 prefix for the SourceIP and TargetIP fields.

Alert data from Warden for 4 months: May, June, July and August 2019, comprising of around 260 million alerts was used for the training and evaluation purposes. Some pertinent statistics related to the 4 month data are presented in Table 2.

4. Objective

The proposed method aims to predict parameters of future alerts related to a given attack source based on information about previous alerts. Since the alert trail from each malicious source is in the form of a sequence, the problem is modeled as a sequence prediction problem.

Recurrent Neural Network architectures have been demonstrated to be suitable for such tasks [54]. Although theoretically capable, conventional RNNs struggle when there are long-term dependencies in the data. LSTMs and GRUs have been proposed as alternatives to the standard RNN and are capable of addressing the vanishing gradient problem in RNNs resulting in improved long-term dependency learning. LSTMs and GRUs have been applied to a variety of sequence classification tasks like Speech Recognition, Semantic Parsing, Human Activity Recognition, etc. With LSTMs and GRUs also proven successful for applications like Time Series Prediction [55] and Time Series Anomaly Detection [56], it is expected that these recurrent networks would also be able to provide high performance solutions to the alert prediction task. This paper therefore considers the use of GRU-based RNNs for forecasting network intrusions. The choice of GRU over LSTM was inspired by the fact that the former have lesser parameters and therefore take lesser time to train and generalize. This was verified during the training process where the use of LSTM resulted in similar prediction performace as compared to GRU, albeit with the requirement of longer training times. For instance, for one of the different cases considered (Scenario 1, in Table 4, the GRU-based model took 4:59 hours to train and yielded prediction accuracy of 69.98% whereas for the same data, an LSTM-based model took 6:12 hours to train, and results in an accuracy of 71.03%).

4.1. Goal Definition

Since the intrusion attacks from individual sources may be treated as sequences, it is deemed prudent to model the alert prediction task as a sequence prediction problem. The input to the machine learning model is a sequence of alerts reporting a malicious source, and the model is trained to output future alerts related to the same source which may occur in the future. Therefore, the goal of the model may be defined as follows: Given a sequence of `History` previous alerts with a particular SourceIP, predict a sequence of next `Future` alerts with the same SourceIP.

The `History` input alerts each comprise of the following fields `DetectTime`, `FlowCount`, `Port`, `Protocol`, `Category`, `Node`, `Node Type`, `TargetIP`, and `SourceIP`. For the purpose of this work, `History` was chosen to be 10. Therefore, the input to the model is a sequence of 10 consecutive alerts with a particular SourceIP.

The output of the model is a sequence of `Future` alerts. Only the fields `DetectTime`, `FlowCount`, `Protocol`, `Port`, `Category` and `TargetIP` are to be predicted on output. That is because we are only interested in parameters of the event itself, not which detector will report it (so `Node`, `Node Type` are missing), and SourceIP is given implicitly. For the proposed work, the value of `Future` was chosen to be 2. Hence, the model outputs 2 possible future alerts for the 10 past alerts from a given SourceIP. Fig. 1 depicts one training vector which shows 10 past alerts and 2 future alerts. The test vectors would only be having the 10 past alerts and the model outputs the 2 future alerts.

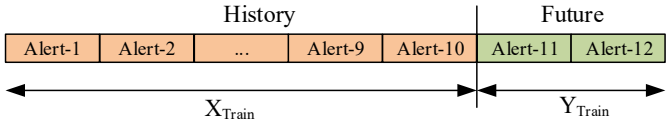

Figure 1: Illustration of a training vector for the case where `History`=10 and `Future`=2

5. Data Pre-processing

Although it is certainly desirable to have an end-to-end learning system, it is infeasible to directly feed the complex IDEA-formatted alerts from the Warden to the deep learning model. Appropriate data transformation and pre-processing are therefore imperative for the model to be effectively employed. These steps are shown in Fig. 2. First, only the entries corresponding to the chosen alert tuple are retained in the dataset and all other columns are discarded. Thereafter, the issue of missing values in the dataset is addressed as follows. For alerts with missing categorical values such as Protocol, Detector, Type, etc., a new dummy category 'Other *column name*' is created for each of the columns in the dataset (e.g 'Other Protocol', 'Other Detector', etc). Moreover, although the `Port` field in the alerts contains numbers, it was considered to treat it as a categorical variable, since port numbers act just as labels, they do not express any numeric quantity. Toward that end, the `Port` field was categorized into 16 categories as follows. The most frequently occurring 15 port numbers in the entire alert sequence for 4 months were treated as independent categories, and all the other less frequent port numbers were categorized as the 'Other Port' category. For the alerts where TargetIP is not present, a dummy 0.0.0.0 address is inserted. After all the missing values have been taken care of, the pre-processing proceeds as follows. The SourceIP and TargetIP entries are trimmed to their /24 prefixes. Since deep learning algorithms require numerical data rather than textual information, all the categorical variables such as Protocol, Detector, Type, etc. are converted to numeric values by using Label Encoding, which is essentially the process of assigning numerical labels to categorical data. Thereafter, a MinMax scaling is performed for each column of the alert dataset. This ensures that all the training data fed to the deep learning network is normalized between 0 and 1, thereby preventing large values from adversely affecting the training process.

This results in individual alert sequence records for each attacking SourceIP which could be considered as a *threat agent* [57]. Lastly, formatted training vectors need to be generated for the processed data obtained from the previous step.

Depending upon the chosen values of `History` and `Future` windows, the data is rearranged in the form of vectors, with each vector comprising `History+Future` alerts. As discussed in the previous section, for this work one training vector would comprise of 12 alerts. A rolling window approach is used for generating the appropriately formatted training vectors. The vectors corresponding to all the SourceIP sequences are then merged and shuffled.

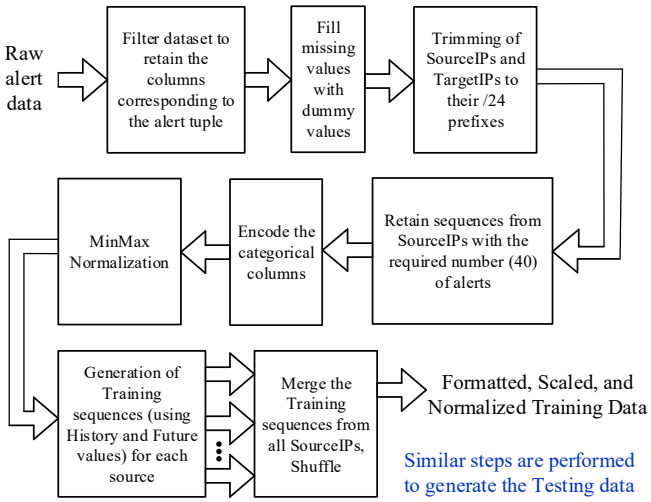

Figure 2: Alert data pre-processing steps

It needs to be mentioned that a particular SourceIP having less than 12 alerts in a calendar month would result in an incomplete training vector. Therefore, alerts from all such SourceIPs were not included in the training dataset. In addition to that, to ensure that a reasonable number of training samples are present for all SourceIPs considered in the dataset, only the SourceIPs reporting 40 or more alerts in a calendar month were included in the training dataset[4].

Similar steps are performed to generate the test dataset. In order to prevent information leakage from the train dataset to the testing phase, it was ensured that the train and test datasets are separated in time. For instance, for one of the test cases (to be elucidated in a subsequent section), the training data comprised of alerts for the month of May–July, and the testing was done on August data. Further, it needs to be mentioned that although Fig. 2 does not show a separate *validation* dataset available at the output of the data pre-processing step, it is indeed a part of the training dataset, and is separated at the time of training by the deep learning library Keras, in proportion to the value specified for the `validation_split` parameter.

6. Proposed GRU-based Deep Learning Network for Alert Prediction

The proposed deep neural network for alert prediction is shown in Fig. 3(a). Training, validation and hyperparameter tuning operations are shown enclosed in the top (red) dashed box, and the test (prediction) operations are shown enclosed in the bottom (blue) dashed box. The details of the model are shown in Fig. 3(b) from where it can be observed that a 3-layer

[4]It is indeed true that the proposed method only works for attackers with higher activity (>40 alerts per month). This is discussed in Section 10. Nevertheless, even if there are <40 alerts from an attacker, it only means we can't *predict* its future behavior, but it still has been detected as malicious (since there are one or more alert(s)). This situation i.e. low rate of attacks is a common way to avoid detection and there are works tackling this problem, but it is out of scope of this work.

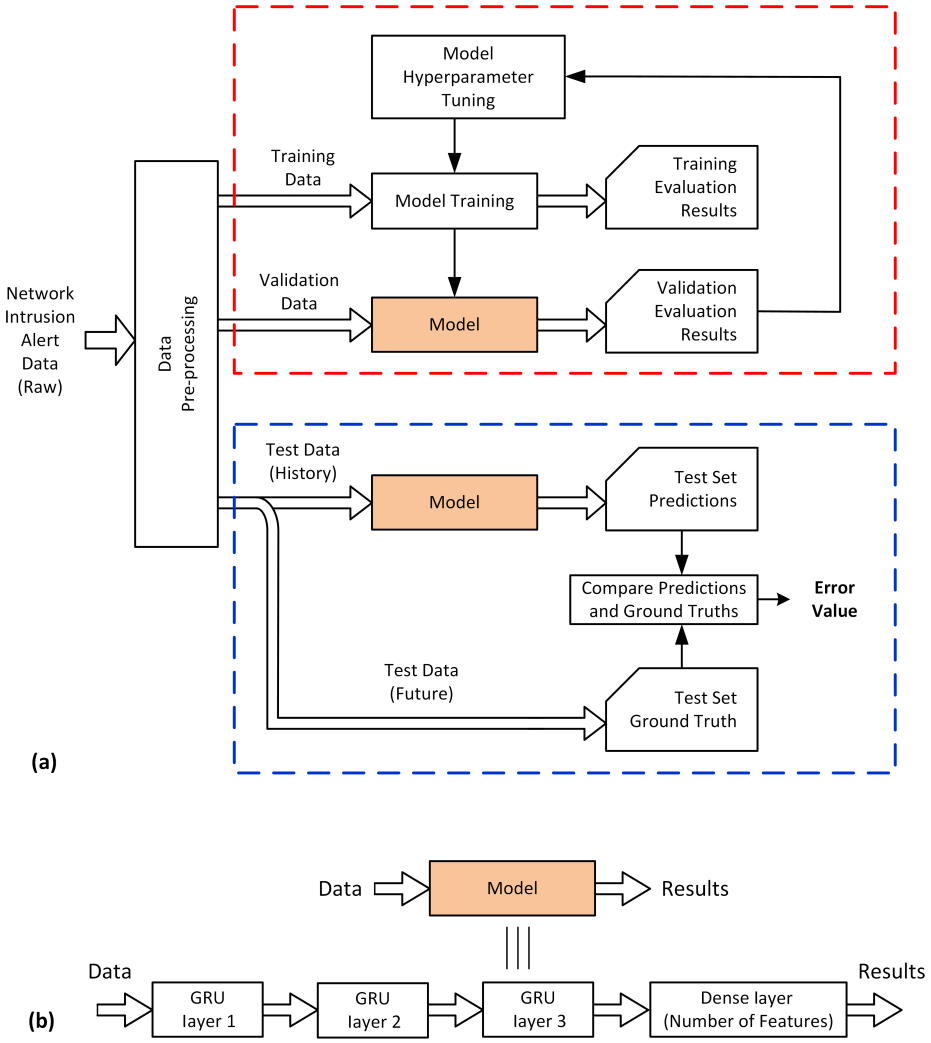

Figure 3: (a) Block diagram of the proposed GRU-based architecture for network intrusion alert prediction. The red dashed block depicts the training and validation processes, and the blue dashed block depicts the testing, *i.e.* making predictions using the trained model and comparing the predictions with the ground truth to obtain the RMSE. (b) Details of the deep learning model

sequential model with GRU layers stacked on the top of a Dense layer is used. Details of the process of parameter selection for the various GRU layers shall be explained in a subsequent section. The number of cells in the Dense layer are equal to the number of features in the data, and for the alert tuple selected for this work, there are 13 features corresponding to one feature each for `DetectTime`, `FlowCount`, `Protocol`, `Node Type`, `Port`, `Category`, `Node`, and three features each for `SourceIP(/24)` and `TargetIP(/24)` obtained after separating the three octets in the respective /24 prefixes.

Raw alert data is first pre-processed according to the steps shown in Fig. 2 and the formatted, scaled and normalized, training and validation datasets are provided to the deep learning model. The training set is used for the training of the deep learning model and the validation set is used to estimate prediction error and model tuning. Once the training, and validation steps are completed, a fine-tuned model is generated. This final model is then used to make predictions on the test dataset.

The proposed network was implemented in Keras which is a high-level neural network API capable of running on top of the deep learning framework TensorFlow™ – an open source software library for high performance numerical computation. For the GRU, the Keras built-in recurrent layer called CuDNNGRU was utilized. It is a fast GRU implementation backed by the Nvidia CUDA® Deep Neural Network library (CuDNN), and can only be run on a Graphical Processing Unit (GPU), with the TensorFlow™ backend. The choice of using GRU over LSTM was inspired by the observation that LSTM-based models took significantly longer to train with almost no improvement in the prediction quality for all the test cases.

The results of the predictions obtained using the model are compared with the ground truth values in the test dataset, and the performance of the model on the predictions is estimated by calculating the accuracy of the predictions. Details of the accuracy estimation process are provided in a subsequent section.

6.1. Parameter Selection

For the purpose of identifying the optimum values of the different configuration parameters of the proposed network, experiments were carried out on a randomized selection of 20% of the total number of training vectors generated for the entire dataset. This implies that the training subset for the parameter selection experiments consisted of 20% of randomly selected vectors from the training dataset generated according to Fig. 2, and the predictions were performed on 20% of data from the entire testing dataset. The following values of different parameters were explored:

- No. of GRU Units: [16, 32, 64, 128, 256, 512, 768, 1024]
- Batch Size: [16, 32, 64, 128, 256, 512, 1024, 2048, 4096]
- No. of Epochs: [200, 500, 1000]

The above parameter space gives 216 different configuration selection possibilities. Training was performed for each of these possible parameter combinations, for the randomly chosen data subset, and the outcome of the training was evaluated on the basis of the Root Mean Square Error (RMSE) calculated by comparing the predictions made on the test dataset using the trained model. It is to be noted that for making and evaluating the predictions using the trained model, the test data subset used was *not* previously seen by the model in either the training or the validation phase. This was done to ensure that there is no information leakage from the training and validation phases to the prediction stage.

To identify the parameters for narrowing down the best performing model, Heat Maps were used. Fig 4 shows two such heat maps obtained as a result of comparing the RMSE for variations across different parameters such as Number of GRU Cells, Batch Size, and epochs. Fig 4(a) depicts the best RMSE values obtained when the number of GRU cells were considered along with the number of epochs. It was seen that training the model(s) for 500 epochs led to the best RMSE values for all the training iterations (shown in shades of green). Increasing the number of epochs beyond 500 led to a degradation in the prediction performance of the model. Fig 4(b) helps to identify a suitable `Batch Size` by comparing the best case RMSEs obtained for all possible combinations of GRU Units and Batch Size. It is evident that Batch Size equal to 2048 in conjunction with 512 GRU cells appears to be the most promising combination. Increasing the `batch size` beyond 2048 did not lift the model skill. To retain parity between the model runs for the 216 different parameter configurations, the GRU weights were initialized using the default Glorot weight initialization offered by TensorFlow.

Similar experiments were carried out to identify the best performing hyperparameters for the GRUs. It was found that the `Adam` optimizer using the default `learning rate` of 0.001, with the `amsgrad` parameter set to `True` which invoked the AMSGrad variant of the Adam algorithm, consistently resulted in smaller RMSE values for most of the test runs. The loss metric which provided the best accuracy was Mean Absolute Error (MAE). The *Early Stopping* feature available in Keras was also invoked to terminate the training process when a certain monitored quantity (chosen to be the 'validation loss' in this case) does not change with further training. A *Patience* value of 10 epochs was selected which means that the Early Stopping was invoked, and the training stopped, whenever the validation loss did not change for ten successive epochs.

7. Estimation of Prediction Accuracy

The quality of predictions is estimated by comparing predicted alerts for all the test vectors with the corresponding ground truth alerts. The dissimilarity metric between actual and predicted alerts (*error value*) is computed as a weighted sum of dissimilarities of individual fields. The comparison method for each field is explained below, and is also depicted in Fig. 5.

Categorical Fields: The categorical fields *viz.* `Category`, `Port`, and `Protocol` are directly compared for the predicted and actual (ground truth) alerts. In case of a match, the error component for that field is set to 0, else it is set to 1. Also, considering that for some ground truth samples, the `Port` and `Protocol` values are 'Other Port' and 'Other Protocol' respectively, meaning that the field was empty in the originally reported alert (and labeled as 'Other' during pre-processing), any prediction for such cases is considered correct, and the error component counted as 0.

TagetIP fields: The 3 octets in the TargetIP field are compared as follows. The first and second octets are required to be exact matches, since any mismatch in these octets would mean that the predicted and actual TargetIPs are very far from each other, probably in a different network altogether. So, in case of mismatch in either the first or second octet, the error component for TargetIP is set to 1.0 and the third octet is not compared. If the first and second octets of the predicted and actual alerts match, then the third octets are compared by computing the number of common leftmost bits, n (length of common prefix in binary notation). The error component for TargetIP is then set to $(1 - (1 + n)/9)$. This causes the error component to smoothly vary from 0.889 ($n = 0$) to 0.0 ($n = 8$).

FlowCount: The error component for the volume of actual and predicted alerts is computed as the difference of the logarithm (base 10) of the `FlowCount` values for the actual and predicted alerts. This corresponds to the human perception by expressing the difference as orders of magnitude. For example, a difference of 10 is not significant when the actual value is 10,000, but it means a large error when the actual value is 1. The result is multiplied by a scaling constant $c_f = 0.5$, so that a 'too large error' (which we consider to be two orders of magnitude here) corresponds to the error value of 1.0.

DetectTime: The method for comparing the `DetectTime` field is based on the same idea as for the the `FlowCount` field – a logarithm was used to ensure that alerts that are expected very soon are considered with higher precision, whereas alerts farther away in the future are accorded lower precision. To achieve this, the absolute timestamps (stored in the `DetectTime` field as the number of seconds from the UNIX epoch) were first recomputed to be relative to the last alert in the history window (shown as $\texttt{diff}_{actual}$ and $\texttt{diff}_{pred}$ in Fig. 5), so they represent how close or far in the future the alerts are. Thereafter, the error component for `DetectTime` was computed as the difference of

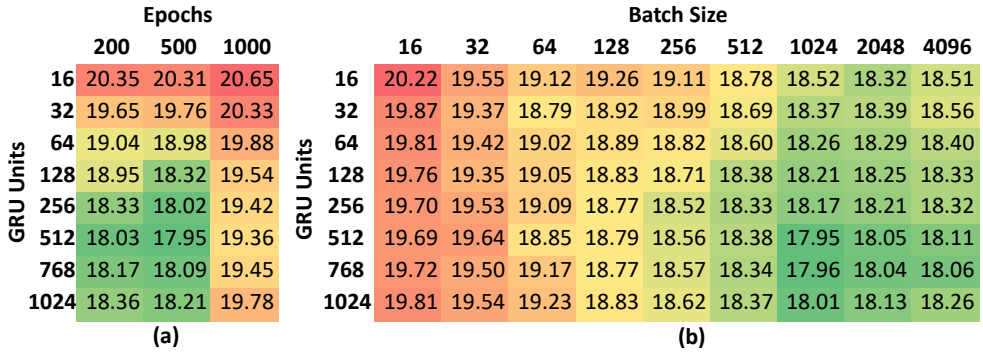

Figure 4: Parameter selection using heat maps. RMSE values shown in the heat maps are for training the model, with randomly selected 20% of the training data, and evaluating on randomly selected 20% of testing data

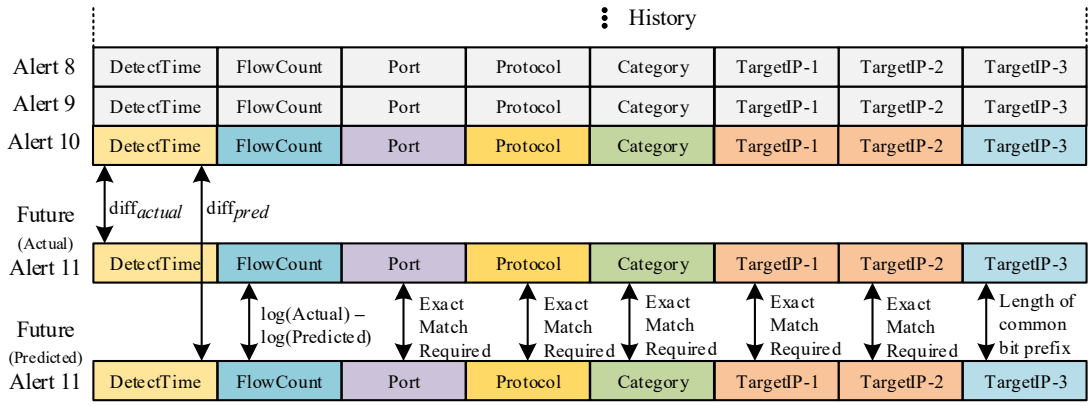

Figure 5: Error estimation process for the predicted alerts

the logarithms of $\texttt{diff}_{actual}$ and $\texttt{diff}_{pred}$. Further scaling was not deemed needed here, so the constant c_d was set to 1.0.

In regard to the numeric fields, it needs to be mentioned that it should not be expected that the predictions shall be able to exactly match their corresponding ground truth counterparts. This is because (i) the `DetectTime` field has timestamp entries which are sometimes separated by minutes, while at other instances are hours apart. This can be attributed to the irregular temporal patterns that attacking sources typically exhibit. (ii) the `FlowCount` field contains the number of connection attempts by an attacking node, and contains entries ranging from one to several thousands. Hence, the task of predicting such a widely varying timestamp or flowcount data can only be expected to yield approximate results.

The error components for the individual fields are then multiplied by weights chosen as per the importance of the respective fields in alert prediction. For instance, error component for `Category` is assigned the highest weight (2.0) since the type of attack is deemed to be the most important characteristic. Approximate time and target (`DetectTime` and `TargetIP` fields) might also be quite important, so they have medium weights (1.0). The `Port` and `Protocol` fields bear similar type of information (the targeted service) are are indeed highly correlated, a mismatch in one often means a mismatch in the other one. Therefore, we assign a weight of 0.5 to each one, so together they have a medium weight of 1.0. The attack volume it not so important in most cases, so the `FlowCount` field gets a low weight (0.5). Table 3 presents a summary of the error estimation process along with all the chosen weights for the error components.

Note that the importance of individual alert fields is quite subjective and therefore the setting of weights is somewhat arbitrary. The weights used in this work were chosen based on practical incident handling experience of one of the authors and his colleagues from a CSIRT team. Others may choose the weights differently and potentially tweak the prediction model to focus more on the fields they see as more important.

The individual error components are then multiplied by corresponding (normalized) weights and summed together. This typically results in most of the error values to lie between 0 and 1, although for very large values of `FlowCount` the error value may exceed unity. However, such cases are rare in the dataset used in this work.

Multiple future alerts are predicted for each test sample (two in the experiments performed in this work). The overall error value is computed as the average of individual one-to-one comparisons of true and predicted alerts. The relative order of predicted alerts is not important during this comparison. This

Table 3: Calculation of field-wise errors and the weights used

	Calculation of Error Component	Weight	Normalized Weight
DetectTime	abs(log(diff$_{actual}$) - log(diff$_{pred}$)) $\cdot c_d$	1.0	0.182
FlowCount	abs(log(actual) - log(predicted)) $\cdot c_f$	0.5	0.091
Port	0 (Match) or 1 (Mismatch)	0.5	0.091
Protocol	0 (Match) or 1 (Mismatch)	0.5	0.091
Category	0 (Match) or 1 (Mismatch)	2.0	0.364
TargetIP-1	0 (Match) or 1 (Mismatch)	1.0	0.182
TargetIP-2	If error for TargetIP-1 = 0: 0 (Match) or 1 (Mismatch)		
TargetIP-3	If error for both TargetIP-1 and TargetIP-2 is 0: $1-(1+n)/9$, where n is the number of common leftmost bits, else: 0		

means that in practice the first predicted alert is compared to both true ones and the one matching it more closely (i.e. lower error value) is used. Then the second predicted alert is compared to the remaining true alert. (Generally, if more than two alerts would be predicted, all permutations would be tried and the best result would be used.) This is not 'cheating', since the `DetectTime` fields still take an important role in the error computation and their difference typically increases with rearrangement, so the original order usually gives the best results. However, in a case where there are two different future alerts (A, B), close in time, and the predicted ones are very similar to them, just in swapped order (B', A'), the overall error value would be very large when compared in the original order (A to B' and B to A'), while it is small in the swapped order – which is correct, since each predicted alert closely matches a real one.

Lastly, in order to get easily interpretable results, we decided to apply some thresholds on the resulting error value and classify the predictions as *good*, *moderate* and *bad*. Based on our experience with the data and behavior of the error metric, we classify results with error value greater than 0.5 as bad predictions – the predicted values differ too much from the real ones to be of any practical use. On the other hand, an error value less than 0.2 means a good prediction since most of the fields are correct or not far from the real values. The values in between are classified as moderate predictions – one of the important fields or multiple less important ones are wrong, but as a whole the predicted alert still resembles the real one.

8. Results

The prediction performance of the proposed GRU-based deep neural network was evaluated for the following scenario: Training on May–July data, testing on August data. As explained in Sec. 6.1, the following set of parameters was identified as the best suited for the task: `Batch Size`: 2048; `No. of GRU Units`: 512; `Epochs`: 500; `Loss parameter`: Mean Absolute Error (MAE); `Optimizer`: Adam with AMSGrad = True; `Patience` (Keras parameter, for Early Stopping): 10.

8.1. Prediction Performance

The prediction performance of the model was estimated by using the error metric explained in the previous section. The model was used to predict future alerts for all the 73,479,259 `History+Future` samples generated from the August 2019 data. The average error for the predictions was found to be 0.130.

Fig. 6 shows a histogram of the error values obtained for the entire test set (shown in red). It can be seen that most of the predictions (76.31%) belong to the 'good' class (error ≤ 0.2). Further, the frequency of alerts with poor predictions (error $>$ 0.5) was observed to be negligible (3.95%) as compared to the well-predicted alerts.

To bring more clarity to the histogram of error values, another histogram with the frequency of errors plotted on a logarithmic scale is also shown as an inset in Fig. 6 (shown in grey). From this log-histogram, it is evident that there are indeed predicted alerts with error values greater than 0.5, albeit in much smaller numbers.

Fig. 7 presents a visual representation of 3 different instances of alerts predicted by the model. Fig. 7(a) presents a case where all the fields are predicted incorrectly. The categorical fields like `Category` and `Protocol` in the actual and predicted alerts do not match, and numeric values like `FlowCount` and `DetectTime` are also off by a large margin. This results in a large error as shown.

Fig. 7(b) shows a moderately good prediction case where all the categorical values are predicted correctly, but the numeric values are not predicted correctly. The predicted `TargetIP` is incorrect, and the `FlowCount` is off by a large margin. However, the predicted `DetectTime` is within one hour of the actual alert. For this prediction example, the error value obtained is 0.36 which is moderately good.

Fig. 7(c) depicts a predicted alert where all the fields, except the `DetectTime` field, are predicted exactly. Even the predicted `DetectTime` value differs from the ground truth value by approximately 30 minutes. The error calculated for this prediction is 1.56e−04.

8.2. Analysis of prediction errors

The errors in the prediction of individual fields of the alerts were analyzed to gain more insight into the model's ability to learn and predict the alerts. Fig. 8 presents the result of this analysis, from where it can be observed that the most difficult field to predict for our model is the `FlowCount` field. On the other hand, the type of attack (`Category` field) turned out to be the most easily predicted value – correct category was predicted in 98 % of cases.

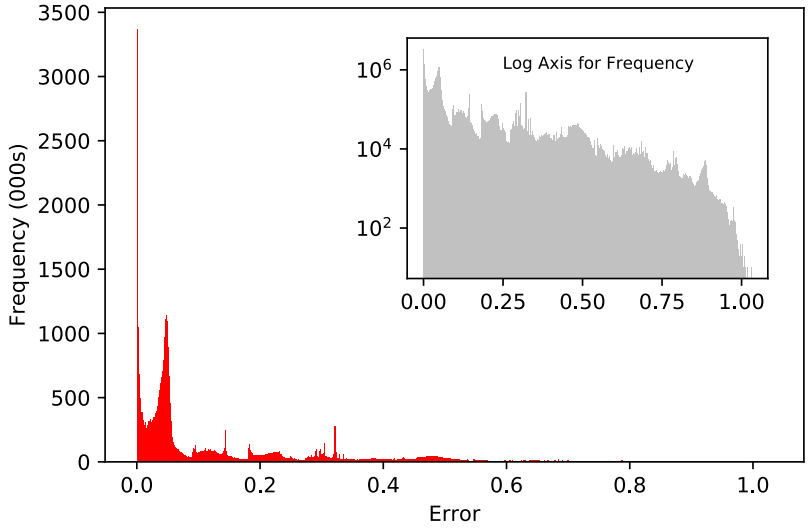

Figure 6: Histogram of error values, training on May–July data, testing on August data. The Y-axis has the frequency of occurrence values in the thousands (denoted by 000s).

	Bad Prediction Example		**Moderate Prediction Example**		**Good Prediction Example**	
	Actual	**Predicted**	**Actual**	**Predicted**	**Actual**	**Predicted**
DetectTime	2019-08-28T18:27:51	2019-08-29T00:32:00	2019-08-11T16:09:19	2019-08-11T16:06:52	2019-08-31T02:03:04	2019-08-31T02:03:08
FlowCount	99	1	10	26	256513	256513
Port	2087	3306	53413	80	22	22
Protocol	TCP	IMAP	TCP	TCP	TCP	TCP
Category	Anomaly.Traffic	Information.UnauthorizedAccess	Recon.Scanning	Recon.Scanning	Attempt.Login	Attempt.Login
TargetIP-1	147	154	87	85	229	229
TargetIP-2	228	236	78	78	213	213
TargetIP-3	190	127	128	130	117	117
Error	**0.96**		**0.36**		**1.56E-04**	
	(a)		**(b)**		**(c)**	

Figure 7: Visual representation of the comparison of actual and predicted alerts for three different cases: (a) Poor match, where most of the fields are predicted incorrectly, and error is very high (b) Average match, where the categorical fields are predicted correctly whereas the numeric fields have a large mismatch (c) Good match, where all categorical fields are predicted correctly, and all numeric fields have good proximity to their actual counterparts.

Further, since the model is designed to learn and predict alert sequences from individual SourceIPs, the number and variety of the alerts (from a particular SourceIP) is also an important factor in the accuracy of predictions. For cases where prediction performance turns out to be poor, the reason may be attributed to the variety in attacks that that particular SourceIP undertakes. This is because a SourceIP choosing to incorporate a large variety into its attacks would generate an alert trail which will be more diverse with an assortment of Port numbers, Protocol values, Category values, *etc.*, and therefore it is harder to find any predictable pattern for the GRU-based model. Conversely, an attacking SourceIP which only uses a limited set of similar attacks becomes a candidate for more accurate predictions. It must also be mentioned that in many cases accurate predictions are just not possible, since the behavior of malicious sources is affected by many factors not known to the prediction model, including such things as random selection of targets in automated scans or attacks.

8.3. Model performance stability in time

As mentioned above, the evaluation results for the proposed DL approach were obtained for the case where the training data comprised of alerts for 3 months (May–July 2019), and testing was performed on August 2019 data. In addition to that, variation in the prediction accuracy over time was explored using 13 other scenarios as shown in Table 4, from where it can be seen that the model performance remains acceptable (average error ≤ 0.2) for all the scenarios considered. In general, training on lesser data leads to poorer prediction results and vice versa.

It can also be noted, especially from Scenario 7, that the system performs acceptably well even when there is a long gap between the months used for training and testing (Scenario 7 was trained on May data, tested on August data). Therefore, the model does not need to be re-trained more often than once in a few months[5].

[5] Assuming no significant change is made to the sources of alert data. For example, when a new detector generating alerts with substantially different characteristics is added, re-training may be necessary sooner.

Table 4: Model performance stability in time

Scenario	May 2019	June 2019	July 2019	August 2019	Average Error	'Good' Predictions (%)
1	Train / Test				0.169	69.98
2		Train / Test			0.152	69.67
3			Train / Test		0.169	65.56
4				Train / Test	0.158	73.77
5	Train	Test			0.153	73.87
6	Train		Test		0.171	70.59
7	Train			Test	0.154	72.25
8		Train	Test		0.132	68.65
9		Train		Test	0.161	68.00
10			Train	Test	0.181	62.22
11	Train		Test		0.168	68.08
12	Train			Test	0.171	67.73
13		Train		Test	0.125	78.96
14	Train			Test	0.130	76.31

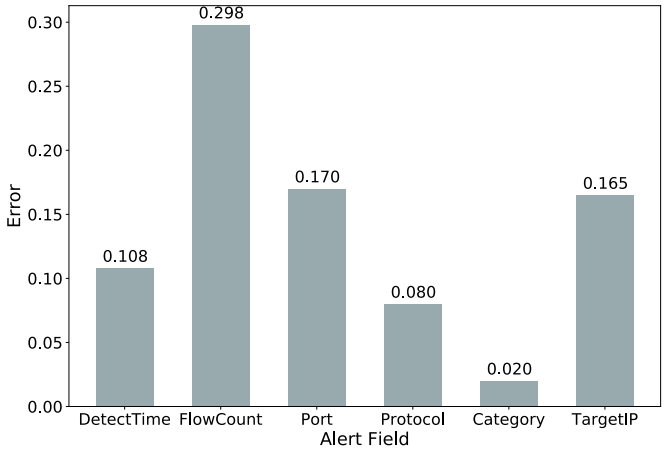

Figure 8: Average error value for each alert field (before weighting)

9. Comparison with Similar Works

This section presents a discussion on comparisons between the proposed approach and related existing works. Early research efforts for prediction of security events mostly culminated in the assignment of probability scores to the next possible security events or alerts, and then identifying the most probable attack scenario [48, 49, 50, 51]. The DL method presented in this work however predicts the entire alert, and therefore cannot be compared directly with the methods in [48, 49, 50, 51].

The two closest works to this paper are DeepLog [42] and Tiresias [8]. DeepLog performs anomaly detection in a regulated environment (such as OpenStack and Hadoop). However, considering the fact that DeepLog focused on only a limited variety of events (40 for OpenStack, 29 for Hadoop) makes it suitable for very specialized log environments. Further, DeepLog's definition of an event's normality is that the event should not be present in the top-g future probable events. Tiresias does away with this relaxed prediction criterion and is able to predict from amongst (upto) 4495 possible events. Further, unlike DeepLog, Tiresias dealt with learning multi-step attacks in a noisy environment. However, Tiresias only predicts the *category* of the next event, which is a *classification* task since it predicts the most probable class out of 4495 classes (event categories). The proposed GRU-based solution, on the other hand, predicts various parameters of the next alert, and since many of them are continuous (such as `DetectTime`, `Flowcount`, etc.), it is a *regression* task. As discussed before, being a regression task for the numeric parameters, the accuracy cannot be expected to compete with the classification based approaches. Another underlying difference between the working of Tiresias (as well as most other alert prediction works) and the proposed method is that while existing solutions predict next attack steps against a particular target, our method predicts next attacks from a particular source. While the former is suitable for hardening targets against future attacks, the latter is more suitable for building an attack profile for malicious sources and is intended to be used as part of threat intelligence platforms, reputation databases, etc.

A notable recent work tackling the issue of predicting next attacks from a given source is the one from Bartos et al. [9], in which authors propose a method of scoring malicious IP addresses by estimating the probability of observing an attack from the IP address in a given future time window. While they can estimate the probability for different attack types separately, the method does not provide any other parameters of the expected attacks. Our method provides this functionality by predicting concrete parameters of future attacks, such as their category, time and target.

Lastly, since there is no available work which predicts exact alert parameters like this paper, it was considered prudent to compare the prediction results with a naive baseline for the data used. Toward that end, each predicted alert was compared with the most-occurring values (for each alert field) in the input sequence of 10 history alerts for all the test samples in Scenario 14. The average error for such a naive baseline test was found to be 0.271 (with 53% well-predicted alerts having error ≤ 0.2), which is significantly more than the error obtained with the trained model (0.130, with 76% well-predicted alerts). Fig. 9 presents a histogram of the error values for the naive baseline test.

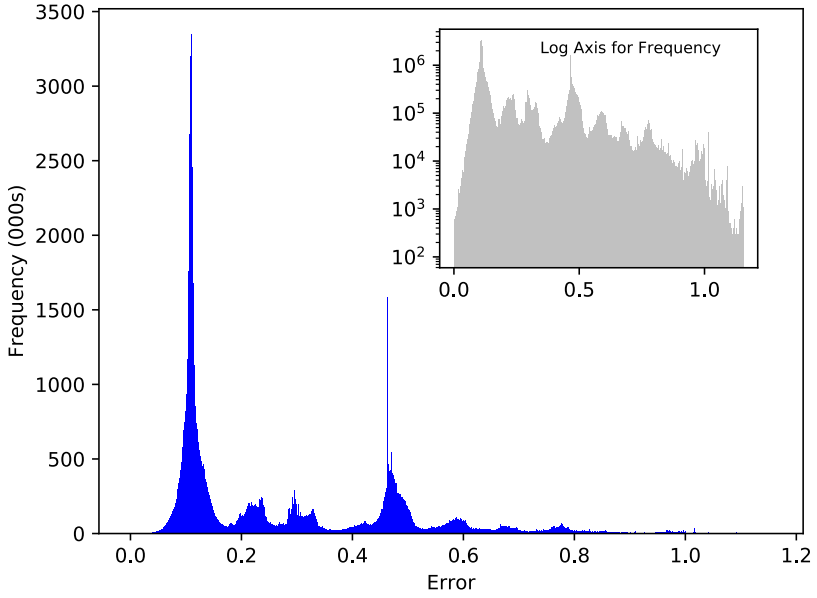

Figure 9: Histogram of error values, for naive baseline test. The Y-axis has the frequency of occurrence values in the thousands (denoted by 000s).

10. Limitations and Future Work

As like with any research attempt, the proposed method is not without its limitations and avenues for further improvements. The identified shortcomings may be attributed to different factors and are discussed below.

Limitations of data: Since the proposed approach is tested on the alert dataset from *Warden* only, it would be interesting to test its performance on security event data from other similar sources. Further, the present study considered data from SourceIPs which had at least 40 alerts per month. This threshold left out information from many 'less frequent' attacking sources. In the future, efforts will be made to get improved performance over the entirety of the data.

Limitations of the analysis: The analysis presented in this work suffers from the following shortcomings. First, only the /24 prefixes of the Source and Target IPs are considered for the analysis. A model working on the complete Source and Target IPs would be more desirable, and this shall be the focus of future endeavors. Second, only the most occurring 15 ports are considered and all the other port numbers are put in the 'Other' category. The number of categories for the ports may be increased in a future extension of this work.

Also, since the proposed model was fine-tuned for the case where `History` and `Future` values were 10 and 2 respectively, the performance of model degrades (as is to be expected) for a different test case where `History` and `Future` are set to 20 and 5 respectively, and the model re-trained on May–Jul 2019 data, and tested on August 2019 data (Scenario 14). In this case, the model generates predictions with an average error of 0.351, with only 18.9% well-predicted alerts. Fig. 10 presents the error histogram for this test, from where it can be observed that model performance is poorer as compared to the case where `History`=10 and `Future`=2.

Limitations of the model: The choice of employing Gated Recurrent Units was inspired by the GRU's inherent ability to learn long sequences, and the fact that training time is better (lower) than LSTMs for obtaining comparable accuracy of predictions. However, other promising contenders for the task include: (i) Convolutional LSTM network[58], which essentially is a variant of Long Short-Term Memory (LSTM) containing a convolution operation inside the LSTM cell, and (ii) Attention networks[59], which are neural networks incorporating the Attention mechanism which equips it with the ability to focus on a subset of its inputs (or features). In fact, Attention networks may be helpful in addressing the inherent issue of the model assigning equal importance to all fields during training, and may be configured to assign higher importance to the correct prediction of the more significant fields in the alert (e.g. `Category`). These shall be explored in subsequent research endeavors.

11. Conclusion

This paper presented a GRU-based deep learning approach for alert prediction. Network intrusion alerts provided by multiple detection systems via a sharing system called Warden were used to train the deep learning model, and subsequently predict future alerts originating from malicious sources. The proposed approach is different from the existing works in the literature in the sense that most of the available works either perform classification on the incoming alerts and assign a label: benign or malicious; or estimate the probability of a particular source IP attacking in a given prediction window. Our method is able

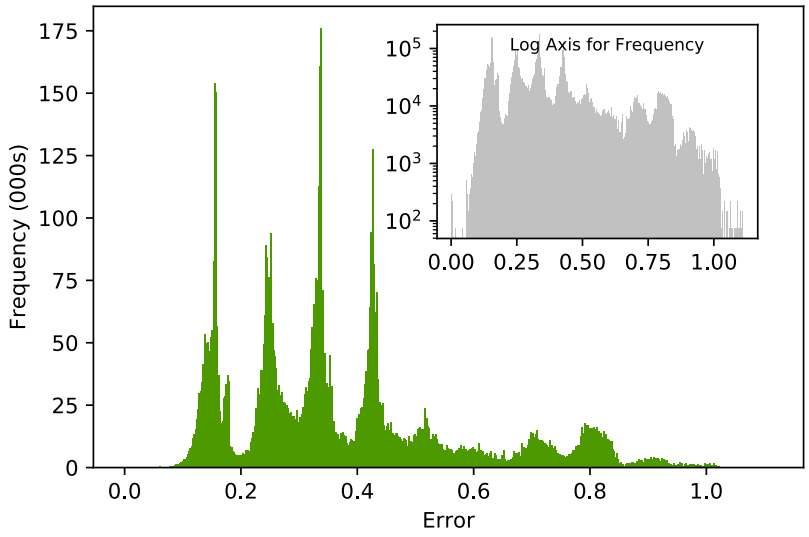

Figure 10: Histogram of error values, for the case where the model fine-tuned for `History`=10, `Future`=2 was re-trained and tested for `History`=20, `Future`=5 (Scenario 14). The Y-axis has the frequency of occurrence values in the thousands (denoted by 000s).

to predict various parameters of the next few attacks coming from a source with acceptable accuracy – over 76 % of predicted alerts have parameters very close to those actually detected.

References

[1] T. D. Wagner, K. Mahbub, E. Palomar, A. E. Abdallah, Cyber threat intelligence sharing: Survey and research directions, Computers & Security 87 (2019).

[2] ENISA, Standards and tools for exchange and processing of actionable information, 2014.

[3] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, M. Fischer, Taxonomy and survey of collaborative intrusion detection, ACM Computing Surveys (CSUR) 47 (2015) 55.

[4] N. Sun, J. Zhang, P. Rimba, S. Gao, L. Y. Zhang, Y. Xiang, Data-driven cybersecurity incident prediction: A survey, IEEE Communications Surveys & Tutorials 21 (2019) 1744–1772.

[5] M. Husák, J. Komárková, E. BouandHarb, P. Čeleda, Survey of attack projection, prediction, and forecasting in cyber security, IEEE Communications Surveys & Tutorials 21 (2019).

[6] M. Husák, V. Bartoš, P. Sokol, A. Gajdoš, Predictive methods in cyber defense: Current experience and research challenges, Future Generation Computer Systems 115 (2021).

[7] P. Sokol, A. Gajdoš, Prediction of attacks against honeynet based on time series modeling, in: Applied Computational Intelligence and Mathematical Methods, Springer International Publishing, Cham, 2018, pp. 360–371.

[8] Y. Shen, E. Mariconti, P. A. Vervier, G. Stringhini, Tiresias: Predicting security events through deep learning, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2018, pp. 592–605.

[9] V. Bartos, M. Zadnik, S. M. Habib, E. Vasilomanolakis, Network entity characterization and attack prediction, Future Generation Computer Systems 97 (2019) 674–686. doi:10.1016/j.future.2019.03.016.

[10] M. Husák, T. Bajtoš, J. Kašpar, E. Bou-Harb, P. Čeleda, Predictive cyber situational awareness and personalized blacklisting: A sequential rule mining approach, ACM Trans. Manage. Inf. Syst. 11 (2020).

[11] J. Zhang, P. Porras, J. Ullrich, Highly predictive blacklisting, in: Proceedings of the 17th Conference on Security Symposium, SS'08, USENIX Association, Berkeley, CA, USA, 2008, pp. 107–122.

[12] F. Soldo, A. Le, A. Markopoulou, Blacklisting recommendation system: Using spatio-temporal patterns to predict future attacks, IEEE Journal on Selected Areas in Communications 29 (2011) 1423–1437.

[13] C. Sauerwein, C. Sillaber, A. Mussmann, R. Breu, Threat intelligence sharing platforms: An exploratory study of software vendors and research perspectives, in: Proceedings der 13. Internationalen Tagung Wirtschaftsinformatik (WI 2017), 2017, pp. 837–851.

[14] Ponemon Institute, Third annual study on exchanging cyber threat intelligence: There has to be a better way, Research Report, 2018. URL: https://ponemonsullivanreport.com/2018/02/third-annual-study-on-exchanging-cyber-threat-intelligence-ther

[15] T. Jansky, T. Cejka, M. Zadnik, V. Bartos, Augmented DDoS Mitigation with Reputation Scores, in: Proceedings of the 13th International Conference on Availability, Reliability and Security, ARES 2018, ACM, New York, NY, USA, 2018, pp. 54:1–54:7. doi:10.1145/3230833.3233279.

[16] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, arXiv preprint arXiv:1406.1078 (2014).

[17] B. Lee, S. Amaresh, C. Green, D. Engels, Comparative study of deep learning models for network intrusion detection, SMU Data Science Review 1 (2018) 8.

[18] G. Kim, H. Yi, J. Lee, Y. Paek, S. Yoon, LSTM-based system-call language modeling and robust ensemble method for designing host-based intrusion detection systems, arXiv preprint arXiv:1611.01726 (2016).

[19] C. Yin, Y. Zhu, J. Fei, X. He, A deep learning approach for intrusion detection using recurrent neural networks, IEEE Access 5 (2017) 21954–21961.

[20] S. J. Yang, H. Du, J. Holsopple, M. Sudit, Attack Projection, Springer International Publishing, Cham, 2014, pp. 239–261.

[21] M. Husák, J. Kašpar, AIDA Framework: Real-Time Correlation and Prediction of Intrusion Detection Alerts, in: Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES 2019), ACM, New York, 2019, pp. "81:1"–"81:8".

[22] K. Zhang, S. Luo, Y. Xin, H. Zhu, Y. Chen, Online mining intrusion patterns from ids alerts, Applied Sciences 10 (2020).

[23] R. Katipally, L. Yang, A. Liu, Attacker behavior analysis in multi-stage attack detection system, in: Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research, CSIIRW '11, Association for Computing Machinery, New York, NY, USA, 2011.

[24] A. Bar, B. Shapira, L. Rokach, M. Unger, Identifying attack propagation patterns in honeypots using markov chains modeling and complex networks analysis, in: 2016 IEEE International Conference on Software

Science, Technology and Engineering (SWSTE), 2016, pp. 28–36.

[25] I. Ghafir, K. G. Kyriakopoulos, S. Lambotharan, F. J. Aparicio-Navarro, B. Assadhan, H. Binsalleeh, D. M. Diab, Hidden markov models and alert correlations for the prediction of advanced persistent threats, IEEE Access 7 (2019) 99508–99520.

[26] I. Perry, L. Li, C. Sweet, S. Su, F. Cheng, S. J. Yang, A. Okutan, Differentiating and predicting cyberattack behaviors using lstm, in: 2018 IEEE Conference on Dependable and Secure Computing (DSC), 2018.

[27] G. Werner, S. Yang, K. McConky, Time series forecasting of cyber attack intensity, in: Proceedings of the 12th Annual Conference on Cyber and Information Security Research, CISRC '17, Association for Computing Machinery, New York, NY, USA, 2017.

[28] H. Wang, D. Zhao, X. Li, Research on network security situation assessment and forecasting technology, Journal of Web Engineering (2020) 264–283.

[29] M. Abdlhamed, K. Kifayat, Q. Shi, W. Hurst, Intrusion Prediction Systems, Springer International Publishing, Cham, 2017, pp. 155–174.

[30] A. Hernández, V. Sanchez, G. Sánchez, H. Pérez, J. Olivares, K. Toscano, M. Nakano, V. Martinez, Security attack prediction based on user sentiment analysis of twitter data, in: 2016 IEEE international conference on industrial technology (ICIT), IEEE, 2016, pp. 610–617.

[31] K. Shu, A. Sliva, J. Sampson, H. Liu, Understanding cyber attack behaviors with sentiment information on social media, in: International Conference on Social Computing, Behavioral-Cultural Modeling and Prediction and Behavior Representation in Modeling and Simulation, Springer, 2018, pp. 377–388.

[32] P. Shao, J. Lu, R. K. Wong, W. Yang, A transparent learning approach for attack prediction based on user behavior analysis, in: International Conference on Information and Communications Security, Springer, 2016, pp. 159–172.

[33] I. A. Gheyas, A. E. Abdallah, Detection and prediction of insider threats to cyber security: a systematic literature review and meta-analysis, Big Data Analytics 1 (2016) 1–29.

[34] Y. Bouzida, F. Cuppens, N. Cuppens-Boulahia, S. Gombault, Efficient intrusion detection using principal component analysis, in: 3éme Conférence sur la Sécurité et Architectures Réseaux (SAR), La Londe, France, 2004, pp. 381–395.

[35] Z. Zhang, H. Shen, Application of online-training svms for real-time intrusion detection with different considerations, Computer Communications 28 (2005) 1428–1442.

[36] Y. Wang, I. Kim, G. Mbateng, S.-Y. Ho, A latent class modeling approach to detect network intrusion, Computer communications 30 (2006) 93–100.

[37] W.-H. Chen, S.-H. Hsu, H.-P. Shen, Application of svm and ann for intrusion detection, Computers & Operations Research 32 (2005) 2617–2634.

[38] O. Depren, M. Topallar, E. Anarim, M. K. Ciliz, An intelligent intrusion detection system (ids) for anomaly and misuse detection in computer networks, Expert systems with Applications 29 (2005) 713–722.

[39] A. Javaid, Q. Niyaz, W. Sun, M. Alam, A deep learning approach for network intrusion detection system, in: Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016, pp. 21–26.

[40] J. Kim, J. Kim, H. L. T. Thu, H. Kim, Long short term memory recurrent neural network classifier for intrusion detection, in: Platform Technology and Service (PlatCon), 2016 International Conference on, IEEE, 2016, pp. 1–5.

[41] S. S. Roy, A. Mallik, R. Gulati, M. S. Obaidat, P. Krishna, A deep learning based artificial neural network approach for intrusion detection, in: International Conference on Mathematics and Computing, Springer, 2017, pp. 44–53.

[42] M. Du, F. Li, G. Zheng, V. Srikumar, Deeplog: Anomaly detection and diagnosis from system logs through deep learning, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2017, pp. 1285–1298.

[43] J. Kim, N. Shin, S. Y. Jo, S. H. Kim, Method of intrusion detection using deep neural network, in: Big Data and Smart Computing (BigComp), 2017 IEEE International Conference on, IEEE, 2017, pp. 313–316.

[44] N. Shone, T. N. Ngoc, V. D. Phai, Q. Shi, A deep learning approach to network intrusion detection, IEEE Transactions on Emerging Topics in Computational Intelligence 2 (2018) 41–50.

[45] S. Aljawarneh, M. Aldwairi, M. B. Yassein, Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model, Journal of Computational Science 25 (2018) 152–160.

[46] J. Zhang, P. A. Porras, J. Ullrich, Highly predictive blacklisting., in: USENIX Security Symposium, 2008, pp. 107–122.

[47] F. Soldo, A. Le, A. Markopoulou, Blacklisting recommendation system: Using spatio-temporal patterns to predict future attacks, IEEE Journal on Selected Areas in Communications 29 (2011) 1423–1437. doi:10.1109/JSAC.2011.110808.

[48] A. A. Ramaki, M. Khosravi-Farmad, A. G. Bafghi, Real time alert correlation and prediction using bayesian networks, in: 2015 12th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC), 2015, pp. 98–103. doi:10.1109/ISCISC.2015.7387905.

[49] A. Okutan, G. Werner, K. McConky, S. J. Yang, Poster: Cyber attack prediction of threats from unconventional resources (capture), in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS17, ACM, New York, NY, USA, 2017, pp. 2563–2565.

[50] P. Holgado, V. A. Villagra, L. Vazquez, Real-time multistep attack prediction based on hidden markov models, IEEE Transactions on Dependable and Secure Computing (2018) 1–1. doi:10.1109/TDSC.2017.2751478.

[51] J. Wang, Y. Yi, H. Zhang, N. Cao, Network attack prediction method based on threat intelligence, in: X. Sun, Z. Pan, E. Bertino (Eds.), Cloud Computing and Security, Springer International Publishing, Cham, 2018, pp. 151–160.

[52] P. Kácha, M. Kostenec, A. Kropácová, Warden 3: Security event exchange redesign, in: Proceedings of the 19th International Conference on Computers: Recent Advances in Computer Science, 2015.

[53] G. Moura, R. Sadre, A. Pras, Bad neighborhoods on the internet, IEEE Communications Magazine 52 (2014) 132–139.

[54] Q. Yang, Z. He, F. Ge, Y. Zhang, Sequence-to-sequence prediction of personal computer software by recurrent neural network, in: 2017 International Joint Conference on Neural Networks (IJCNN), IEEE, 2017, pp. 934–940.

[55] J. Schmidhuber, D. Wierstra, F. J. Gomez, Evolino: Hybrid neuroevolution/optimal linear search for sequence prediction, in: Proceedings of the 19th International Joint Conferenceon Artificial Intelligence (IJCAI), 2005.

[56] P. Malhotra, L. Vig, G. Shroff, P. Agarwal, Long short term memory networks for anomaly detection in time series, in: Proceedings, Presses universitaires de Louvain, 2015, p. 89.

[57] T. Bajtoš, A. Gajdoš, L. Kleinová, K. Lučivjanská, P. Sokol, Network intrusion detection with threat agent profiling, Security and Communication Networks 2018 (2018).

[58] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, W.-c. Woo, Convolutional lstm network: A machine learning approach for precipitation nowcasting, in: Advances in neural information processing systems, 2015, pp. 802–810.

[59] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems, 2017, pp. 5998–6008.