Detecting Network Anomalies and Intrusions in Communication Networks

Ana Laura Gonzalez Rios, Zhida Li, Guangyu Xu, Alfonso Diaz Alonso, and Ljiljana Trajković
Simon Fraser University
Vancouver, British Columbia, Canada
Email: {anag, zhidal, gxa5, adiazalo, ljilja}@sfu.ca

***Abstract*—Detecting anomalies and intrusions in communication networks is of great interest in cyber security. In this paper, we use Support Vector Machine (SVM) and Broad Learning System (BLS) supervised machine learning approaches to detect anomalies and intrusions in datasets collected from packet data networks. The developed models are trained and tested using data from the Internet routing tables, a simulated air force base network, and an experimental testbed. These datasets contain records of both intrusions and regular traffic data. We compare the two machine learning algorithms based on accuracy, F-Score, and training time.**

***Keywords*—Machine learning, support vector machine, broad learning system, anomaly and intrusion detection**

I. Introduction

The Internet is highly susceptible to failures and attacks. Malicious activities in communication networks are monitored using various intrusion detection systems (IDSes) [13]. They are important for identifying and classifying anomalies and intrusions. Most IDSes are behavior-based where regular patterns in network traffic are defined and then traffic is scanned for non-conforming patterns [26]. They may be network-based or host-based. Network-based systems monitor incoming network traffic (Internet Protocol addresses, service ports, and protocols) while host-based systems monitor operating system files and processes. Various machine learning models have been implemented to enhance cyber security [22].

Support Vector Machine (SVM) [17], Recurrent Neural Networks (RNNs) such as Long Short-Term Memory (LSTM) [23] and Gated Recurrent Unit (GRU) [16], and Broad Learning System (BLS) [14], [15], [28] supervised machine learning models have been used to classify known network anomalies and intrusions [25]. These models have demonstrated robustness, high accuracy, and short training time when classifying Border Gateway Protocol (BGP) anomalies [11], [18], [19], [24]. LSTM, GRU, and BLS models also successfully identified anomalies when using the NSL-KDD dataset [24].

Reliable identification, testing, and validation of anomalies and intrusions depend on the quality of datasets. Anomaly and intrusion detection systems may be evaluated using traffic collected from deployed networks or experimental testbeds. The developed models are evaluated using data collected from Réseaux IP Européens (RIPE) [1], the NSL-KDD benchmark dataset [2], and the CICIDS2017 data collected by the Canadian Institute for Cybersecurity (CIC) [3].

RIPE data contain messages collected from BGP routing tables. BGP is an incremental path vector Internet routing protocol that distributes notifications about the changes in topologies and reachability through update and withdrawal messages. BGP manages network reachability information and selects the most preferable routes using four types of messages: *open*, *update*, *notification*, and *keepalive*. BGP relies on the Transmission Control Protocol (TCP) for exchanging messages between routers. BGP is vulnerable and prone to anomalies that impede the successful reachability message exchange and may generate thousands of anomalous BGP updates [10], [32]. Several proposals and modifications have been introduced to improve BGP security [20], [29].

Deployed testbeds consist of firewalls, routers, switches, and operating systems that are simulated, emulated, or built with physical network elements such as: National Cyber Range (NCR), Emulab, Testbed@TWISC, DETERlab, PlanetLab, Self-organising Adaptive Technology Underlying Resilient Networks (SATURN), StarBED, and Global Environment for Network Innovations (GENI) [21].

NSL-KDD dataset consists of selected features from the KDD'99 dataset that contained TCP, User Datagram Protocol (UDP), and Internet Control Message Protocol (ICMP) traffic collected using the *tcpdump* utility. Data were captured from an evaluation testbed simulating a small air force base and included large number of virtual hosts and user automata [27].

CIC developed a testbed framework that generated regular (benign) and intrusion (attack) traffic data [30]. Based on this framework, a victim-network and an attacker-network were implemented to generate CICIDS2017 dataset [31]. The victim-network testbed included three servers, one firewall, two switches, and ten PCs interconnected by a domain controller and active directory. One switch was configured as a mirror port to capture the network traffic. The attacker-network consisted of one router, one switch, and four PCs.

This paper is organized as follows: We describe BGP, NSL-KDD, and CICIDS2017 datasets in Section II. The Support Vector Machine and BLS models are presented in Section III and Section IV, respectively. The experimental procedure and training parameters employed as well as their performance measures are given in Section V. We conclude with Section VI.

This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada under grant R31-611284.

II. Description of Datasets

We consider regular (0) and anomalous (1) classes in BGP, NSL-KDD, and CICIDS2017 datasets.

A. BGP Datasets

BGP datasets were extracted from BGP update messages collected from RIPE [1]. In this paper, we analyze three well-known anomalies: Slammer, Nimda, and Code Red I as shown in Table I. We consider 37 numerical features. Each dataset contains 7,200 data points, obtained from five days of anomalous and regular data. Hence, each data point represents one minute of routing records.

TABLE I
BGP Internet Anomalies

Dataset	**Beginning of event** GMT	**End of event** GMT	**Duration** (min)
Slammer	25.01.2003, 05:31	25.01.2003, 19:59	869
Nimda	18.09.2001, 13:19	20.09.2001, 23:59	3,521
Code Red I	19.07.2001, 13:20	19.07.2001, 23:19	600

During the Slammer attack [5], Microsoft Structured Query Language (SQL) servers and PCs with Microsoft SQL Server Data Engine (MSDE) were infected through a code that randomly generated IP addresses. The code replicated itself by infecting new machines through randomly generated targets. The number of infected machines doubled approximately every 9 s.

Nimda attack [6] exploited vulnerabilities in the Microsoft Internet Information Services (IIS) web servers for the Internet Explorer 5. The worm propagated by sending an infected attachment that was automatically downloaded after viewing email. A user could also download it from the website or access an infected file through the network. Nimda infected over 2.2 million servers and PCs in a 24-hour period.

The Code Red I worm [7] attacked Microsoft IIS web servers, replicated itself by exploiting weakness of the IIS servers, and searched for vulnerable servers to infect. Rate of infection doubled every 37 min during the attack.

Concatenations of two datasets are used for training while the third dataset is used for testing as shown in Table II.

TABLE II
BGP Training and Test Datasets

Dataset	**Training dataset**	**Test dataset**
S+N$\rightarrow$C	Slammer and Nimda	Code Red I
S+C$\rightarrow$N	Slammer and Code Red I	Nimda
N+C$\rightarrow$S	Nimda and Code Red I	Slammer
N+S$\rightarrow$C	Nimda and Slammer	Code Red I
C+S$\rightarrow$N	Code Red I and Slammer	Nimda
C+N$\rightarrow$S	Code Red I and Nimda	Slammer

We also partition the datasets by selecting 80 %, 70 %, or 60 % of anomalous data for training and the remaining 20 %, 30 %, or 40 % for testing. (We keep the number of data points in the training and test datasets to be divisible by 20 by moving the remaining data points from the training to the test dataset.) The number of data points in the BGP training and test datasets are listed in Table III.

B. NSL-KDD Dataset

NSL-KDD [2] is an improved version of the KDD'99 intrusion dataset, which is based on DARPA 1998 dataset. It is a benchmark widely used for evaluation of anomaly detection techniques. The NSL-KDD dataset is a randomly selected subset of KDD'99 after redundant data were removed. It contains four types of attacks: denial of service (DoS), user to root (U2R), remote to local (R2L), and probe.

NSL-KDD dataset contains nine weeks of collected traffic when various intrusions were introduced in a simulated air force base network [27]. Each network connection is represented by 41 features. The dataset contains one training (KDDTrain$^+$) and two test datasets (KDDTest$^+$ and KDDTest^{-21}). KDDTest^{-21} is a subset of the KDDTest$^+$ dataset that does not include records correctly classified by 21 models [33]. The number of data points is shown in Table IV.

C. CICIDS2017 Dataset

The CIC testbed is used to create the publicly available CICIDS2017 dataset that includes multiple types of recent cyber attacks. Network traffic was collected between Monday, 03.07.2017 at 09:00 and Friday, 07.07.2017 at 17:00. The dataset files contain raw *pcap* and extracted data that include features and labels for each flow. The extracted data are based on TCP, UDP, and IPv6 hop-by-hop protocol flows. They are generated using a testbed having both the victim and the attacker networks. Regular traffic captures behavior of 25 users and includes data based on HTTP, HTTPS, FTP, SSH, and email protocols (SMTP, POP3, and IMAP). The testbed for CICIDS2017 contains interconnected Windows workstations including service packs with various known vulnerabilities and Linux based machines with the Metasploit-defined distribution designed for attacks by penetration testers.

Traffic flows collected on Monday, 03.07.2017 are regular while eight types of intrusion attacks have been initiated between Tuesday, 04.07.2017 and Friday, 07.07.2017: brute force using file transfer protocol (FTP) and secure shell (SSH), heartbleed, web attack, infiltration, botnet, denial of service (DoS), and distributed denial of service (DDoS) [3], [4]. List of intrusion attacks and number of flows are shown in Table V and Table VI, respectively. These intrusions rely on various network vulnerabilities [31]. Brute force is used for password guessing through repetitive attempts. Heartbleed attacks a network by accessing the memory of the system using Open Secure Sockets Layer (OpenSSL) cryptography library. Web attack involves SQL injection, cross-site scripting, and brute force using HTTP. Infiltration exploits vulnerable software and are conducted from inside network. The botnet attackers control devices and perform various attacks through the Internet. DoS and DDoS make resources of target machines unavailable due to flooding.

Intrusions were executed using malicious attack tools such as Patator, Slowloris, Heartleech, Damn Vulnerable Web App, Metasploit, Ares, and Low Orbit Ion Cannon. The publicly available CICFlowMeter [4] tool was employed to extract

TABLE III
NUMBER OF DATA POINTS FOR BGP TRAINING AND TEST DATASETS

	Partition (%) (training/testing)	**Regular** (training)	**Anomalies** (training)	**Regular** (test)	**Anomalies** (test)	**Beginning of collection**	**End of collection**
Slammer	60/40	3,740	531	3,460	339	23.01.2001 00:00:00	27.01.2001 23:59:59
	70/30	3,820	611	3,380	259		
	80/20	3,900	691	3,300	179		
Nimda	60/40	5,800	2,123	1,400	1,399	16.09.2001 00:00:00	20.09.2001 23:59:59
	70/30	6,140	2,463	1,060	1,059		
	80/20	6,500	2,823	700	699		
Code Red I	60/40	4,040	362	3,160	239	17.07.2001 00:00:00	21.07.2001 23:59:59
	70/30	4,100	422	3,100	179		
	80/20	4,160	482	3,040	119		

TABLE IV
NSL-KDD DATASET: NUMBER OF DATA POINTS

	Regular	**DoS**	**U2R**	**R2L**	**Probe**	**Total**
KDDTrain$^+$	67,343	45,927	52	995	11,656	125,973
KDDTest$^+$	9,711	7,458	200	2,754	2,421	22,544
KDDTest^{-21}	2,152	4,342	200	2,754	2,402	11,850

84 features. We use the dataset where features *flow ID*, *source IP address*, *source port number*, *destination IP address*, *protocol*, and *timestamp* have been removed from datasets before training and testing. A total of 2,830,743 data points were collected during the five days. We extract data points corresponding to the morning (08:45) and early afternoon (13:00) on Tuesday, 04.07.2017 to construct the training and test datasets. We used this smaller dataset because of the SVM long training time.

III. SUPPORT VECTOR MACHINE

The SVM algorithm is a supervised learning approach used to identify optimal hyperplanes (decision boundaries) [17]. Hyperplanes are optimized by maximizing the margin (minimum distance) between the data points of different classes. The data points that determine the margin are known as support vectors. SVM is a powerful approach for signal and image processing as well as classification of network anomalies. Linearly and nonlinearly separable data may be classified employing SVM. Kernels are linear and nonlinear maps that project the input data points from lower to higher dimensional spaces in order to reduce computational complexity of the algorithm. Examples of popular kernels are: polynomial, Gaussian radial basis function (RBF), and sigmoid. SVM with kernels may be applied to datasets having large number of features without additional computational complexity [34].

Polynomial kernels are defined as:

$$k(\mathrm{x}_i, \mathrm{x}_j) = (\mathrm{x}_i, \mathrm{x}_j + c)^p, \tag{1}$$

where c is the soft margin constant [12] and p determines the degree of the polynomial. These parameters define the error tolerance and shape of the decision boundary of the resulting classifier. The linear kernel ($p = 1$) is useful in case of a large number of features due to its short running time. The quadratic kernel ($p = 2$) maps the data points using a quadratic function that produces an elliptical boundary when cut by a plane.

The SVM kernel using Gaussian RBF is defined as:

$$k(\mathrm{x}_i, \mathrm{x}_j) = \exp\left(-\frac{\|\, x_i \;-\; x_j \,\|^2}{\sigma^2}\right), \tag{2}$$

where σ determines the width of the Gaussian distribution as well as the flexibility and smoothness of the decision boundary. The decision boundaries of Gaussian RBF kernels have the shape of hyperellipses.

The sigmoid kernel is defined by the hyperbolic tangent function:

$$k(\mathrm{x}_i, \mathrm{x}_j) = \tanh(\beta \mathrm{x}_i^{\mathrm{T}} \mathrm{x}_j + r), \tag{3}$$

where β scales the input data and r is a shifting parameter that controls the mapping threshold.

IV. BROAD LEARNING SYSTEM

Deep learning neural networks usually rely on a large number of hidden layers that may employ RNNs such as LSTM and GRU. In contrast, BLS framework [9], [15], an alternative to deep learning networks, improves the structure of a random vector functional-link neural network by mapping the input data $\boldsymbol{X}$ to a set of mapped features $\boldsymbol{Z}^n$ as shown in Fig. 1. Enhancement nodes $\boldsymbol{H}^m$ are generated from these mapped features using random weights.

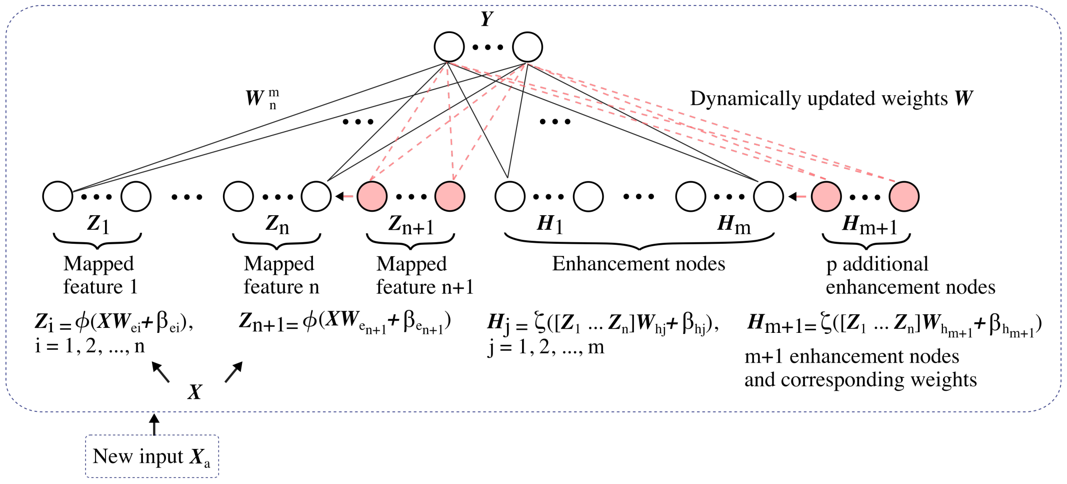
The groups of mapped features and enhancement nodes are defined as:

$$\boldsymbol{Z}_n = \phi(\boldsymbol{X}\boldsymbol{W}_{ei} + \boldsymbol{\beta}_{ei}) \; i = 1, 2, ..., n, \tag{4}$$

$$\boldsymbol{H}_m = \xi(\boldsymbol{Z}_x^n \boldsymbol{W}_{hj} + \boldsymbol{\beta}_{hj}) \; j = 1, 2, ..., m, \tag{5}$$

where ϕ and ξ are the feature and enhancement mappings, respectively. $\boldsymbol{W}_{ei}$ and $\boldsymbol{W}_{hj}$ are weights while $\boldsymbol{\beta}_{ei}$ and $\boldsymbol{\beta}_{hj}$ are bias parameters.

BLS offers comparable performance to deep learning models and has shorter training time [24] when used with large datasets such as NSL-KDD. BLS variations include: BLS with Radial Basis Function (BLS-RBF) [28], BLS with cascades, and incremental BLS. The RBF network has one input, one hidden, and one output layer. It uses Gaussian function as the RBF kernel (2). The cascades of BLS implement various algorithms to create groups of mapped features (CFBLS) and enhancement nodes (CEBLS) [14]. When cascading only mapped features or only enhancement nodes, each new group of mapped features or enhancement nodes is generated using



Y
W n m
Dynamically updated weights W
Z1
Zn
Zn+1
H1
Hm
Hm+1
Mapped feature 1
Mapped feature n
Mapped feature n+1
Enhancement nodes
p additional enhancement nodes
Zi = φ(XWei+βei), i = 1, 2, ..., n
Zn+1= φ(XWen+1+βen+1)
Hj = ζ([Z1 ... Zn]Whj+βhj), j = 1, 2, ..., m
Hm+1= ζ([Z1 ... Zn]Whm+1+βhm+1)
m+1 enhancement nodes and corresponding weights
X
New input Xa

Fig. 1. Module of the BLS algorithm with increments of mapped features, enhancement nodes, and new input data [15].

TABLE V
CICIDS2017 Dataset: Types of Intrusion Attacks

Attack	Label	Day	Number of intrusions
Brute force	FTP, SSH	Tuesday	7,935; 5,897
Heartbleed	Heartbleed	Wednesday	11
Web attack	Brute force, XSS, SQL Injection	Thursday morning	1,507; 652; 21
Infiltration	Infiltration, PortScan	Thursday and Friday afternoons	36; 158,930
Botnet	Bot	Friday morning	1,956
DoS	Slowloris, Hulk, GoldenEye, SlowHTTPTest	Wednesday	5,796; 230,124; 10,293; 5,499
DDoS	DDoS	Friday afternoon	128,027

TABLE VI
CICIDS2017 Dataset: Number of Flows

Day	Valid flows	Total
Monday	529,481	529,918
Tuesday	445,645	445,909
Wednesday	691,406	692,703
Thursday (morning)	170,231	170,366
Thursday (afternoon)	288,395	288,602
Friday (morning)	190,911	191,033
Friday (afternoon, PortScan)	286,096	286,467
Friday (afternoon, DDoS)	225,711	225,745

the previous group. The first group of CFBLS mapped features is created using the input data with $n = 1$ (4). The first CEBLS enhancement node is created using mapped features with $m = 1$ (5). The subsequent groups of mapped features or enhancement nodes are defined as:

$$\begin{aligned} \boldsymbol{Z}_k &= \phi(\boldsymbol{Z}_{k-1}\boldsymbol{W}_{ek} + \boldsymbol{\beta}_{ek}) \\ &\triangleq \phi^k(\boldsymbol{X}\ ;\ \{\boldsymbol{W}_{ei}, \boldsymbol{\beta}_{ei}\ \}\ _{i=1}^{k}), \text{for } k = 1, ..., n, \end{aligned} \quad (6)$$

$$\boldsymbol{H}_u \triangleq \xi^u(\boldsymbol{Z}^n\ ;\ \{\boldsymbol{W}_{hi}, \boldsymbol{\beta}_{hi}\}_{i=1}^{u}),\ for\ u = 1, ..., m. \quad (7)$$

The combination of these cascades results in the cascade of mapped features and enhancement nodes (CFEBLS). Incremental BLS consists of dynamically added mapped features, enhancement nodes, and input data. In this study, we consider only incremental BLS and RBF-BLS without cascades.

V. Experimental Procedure and Performance Evaluation

We evaluate the performance of SVM and BLS using BGP, NSL-KDD, and CICIDS2017 datasets based on accuracy, F-Score, and training time. Since SVM performance is influenced by the kernel function, we compare the algorithm using polynomial (linear, quadratic, and cubic), Gaussian, and sigmoid kernels. BLS architecture as well as the number of mapped features, enhancement nodes, and mapped groups affects the BLS classification performance.

The experimental procedure consists of six steps:

- *Step 1*: Create the BGP, NSL-KDD, and CICIDS2017 training and test datasets.
- *Step 2*: Verify the number of regular and anomalous data points in training and test datasets.
- *Step 3*: Normalize training and test datasets to have mean 0 and variance 1.
- *Step 4*: Train and tune parameters for SVM and BLS models using 10-fold validation.
- *Step 5*: Test SVM and BLS models using BGP, NSL-KDD, and CICIDS2017 datasets.
- *Step 6*: Evaluate derived models based on accuracy and F-Score, and training time.

Experiments are performed using Dell Alienware Aurora with 32 GB memory, NVIDIA GeForce GTX 1080 GPU, and Intel Core i7 7700K processor.

SVM and BLS results are generated using MATLAB R2018b. SVM models and labels for the test dataset are generated using MATLAB functions *fitcsvm* and *predict* [8], respectively. The training data and their labels are stored as matrices. Function *fitcsvm* returns a predictor dataset (the SVM model) based on parameters such as kernel, kernel scale, and penalty factor (box constraint), training matrices, and binary classification. Test data points are labeled using function *predict*, the created SVM model, and test data points. The sigmoid function in the RBF-BLS model is replaced with the RBF function for enhancement nodes. We implement the CFBLS, CEBLS, and CFEBLS models by modifying the original BLS functions [9]. The SVM and BLS training parameters that generate the best performance results are listed in Tables VII, VIII, IX, and X while performance results are given in Tables XI, XII, and XIII.

TABLE VII
SVM TRAINING PARAMETERS USING BGP, NSL-KDD, AND CICIDS2017 DATASETS

Dataset	**Kernel**	**Kernel scale**	**Penalty factor**
S+N→C	Cubic	Default	Default
S+C→N	Sigmoid	Default	Default
N+S→C	Quadratic	Default	Default
N+C→S	Linear	5	90
C+S→N	Cubic	5	90
C+N→S	Quadratic	Default	Default
Slammer (60 %)	Quadratic	Default	Default
Nimda (80 %)	RBF	Default	Default
Code Red I (60 %)	RBF	15	100
NSL-KDD	Cubic	Default	Default
CICIDS2017	Quadratic	5	90

TABLE VIII
BLS TRAINING PARAMETERS USING BGP DATASETS

Parameters	**S+N→C**	**N+C→S**	**C+S→N**
Model	RBF-BLS	RBF-BLS	CFBLS
Mapped features	100	100	300
Groups of mapped features	100	100	100
Enhancement nodes	1,000	1,000	1,000
	Slammer (80 %)	**Nimda (60 %)**	**Code Red I (60 %)**
Model	BLS	RBF-BLS	CFBLS
Mapped features	100	100	300
Groups of mapped features	100	300	100
Enhancement nodes	1,000	1,000	1,000

TABLE IX
INCREMENTAL BLS TRAINING PARAMETERS USING BGP DATASETS

Parameters	**N+S→C**	**S+C→N**	**N+C→S**
Model	RBF-BLS	BLS	RBF-BLS
Mapped features	200	100	100
Groups of mapped features	100	200	200
Enhancement nodes	1,000	100	1,000
Incremental learning steps	5	5	5
Data points/step	1,000	1,000	1,000
Enhancement nodes/step	100	100	100
	Slammer (80 %)	**Nimda (60 %)**	**Code Red I (60 %)**
Model	RBF-BLS	RBF-BLS	BLS
Mapped features	200	100	100
Groups of mapped features	100	200	100
Enhancement nodes	1,000	1,000	1,000
Incremental learning steps	6	9	7
Data points/step	200	200	200
Enhancement nodes/step	100	100	100

TABLE X
BLS AND INCREMENTAL BLS TRAINING PARAMETERS USING NSL-KDD AND CICIDS2017 DATASETS

Parameters	**NSL-KDD**	**CICIDS2017**
BLS		
Model	BLS	CEBLS
Mapped features	100	10
Groups of mapped features	100	20
Enhancement nodes	200	200
Incremental BLS		
Mapped features	100	10
Groups of mapped features	5	20
Enhancement nodes	100	200
Incremental learning steps	3	5
Data points/step	3,000	3,000
Enhancement nodes/step	60	10
Model	BLS	BLS

TABLE XI
PERFORMANCE OF SVM MODELS USING BGP, NSL-KDD, AND CICIDS2017 DATASETS

Dataset	**Accuracy (%)**	**F-Score (%)**	**Kernel**	**Training time (s)**
S+N→C	91.68	2.28	Cubic	118.42
S+C→N	60.47	37.53	Sigmoid	3.26
N+S→C	81.71	42.81	Quadratic	188.58
N+C→S	93.79	77.37	Linear	106.70
C+S→N	54.26	13.68	Cubic	56.89
C+N→S	93.93	77.78	Quadratic	210.20
Slammer (60 %)	88.90	60.41	Quadratic	0.41
Nimda (80 %)	73.14	84.49	RBF	1.10
Code Red I (60 %)	96.65	79.38	RBF	47.45
KDDTest^{+}	96.60	98.27	Cubic	2,128.98
KDDTest^{-21}	99.25	99.62	Cubic	2,131.03
CICIDS2017	99.78	99.89	Quadratic	1.73

VI. CONCLUSION

In this paper, we evaluated accuracy, F-Score, and training time of SVM and BLS algorithms using BGP, NSL-KDD, and CICIDS2017 datasets. Performance evaluation was conducted using concatenation and partitioning of BGP datasets. While the order of concatenations when creating training datasets affects the SVM and incremental BLS accuracy and F-Score, it does not influence performance of BLS models without incremental learning. Only a subset of the CICIDS2017 dataset was used to accommodate longer SVM training time. Experimental results indicate that for NSL-KDD dataset, SVM exhibits significantly higher accuracy and F-Score than BLS. Training time for BLS is much shorter than for SVM when using larger datasets. BLS with cascades of enhancement nodes requires significantly longer training time than other BLS variants. The training time for larger datasets is significantly shorter for incremental BLS because its weights are dynamically updated and, hence, the model does not need to be retrained. Larger number of mapped features and enhancement nodes requires additional memory and longer training time.

TABLE XII
PERFORMANCE OF BLS AND INCREMENTAL BLS MODELS USING BGP DATASETS

Dataset	**Accuracy (%)**	**F-Score (%)**	**Model**	**Training time (s)**
BLS				
S+N→C	80.47	41.22	RBF-BLS	2.88
N+C→S	93.46	77.37	RBF-BLS	2.97
C+S→N	60.47	43.58	CFBLS	6.53
Slammer (80 %)	70.27	82.54	BLS	6.56
Nimda (60 %)	97.86	98.92	RBF-BLS	76.68
Code Red I (60 %)	66.65	79.98	CFEBLS	70.81
Incremental BLS				
N+S→C	91.07	45.09	RBF-BLS	178.53
N+C→S	92.29	72.01	RBF-BLS	178.59
S+C→N	58.65	41.22	BLS	177.86
Slammer (80 %)	89.91	34.58	BLS	125.79
Nimda (60 %)	85.71	92.30	RBF-BLS	193.74
Code Red I (60 %)	89.87	61.88	BLS	33.78

TABLE XIII
PERFORMANCE OF BLS AND INCREMENTAL BLS MODELS USING NSL-KDD AND CICIDS2017 DATASETS

Dataset	**Accuracy (%)**	**F-Score (%)**	**Model**	**Training time (s)**
BLS				
KDDTest^{+}	82.82	82.85	BLS	105.33
KDDTest^{-21}	66.08	74.95	BLS	94.10
CICIDS2017	99.52	93.68	CEBLS	4.08
Incremental BLS				
KDDTest^{+}	81.34	81.99	BLS	32.99
KDDTest^{-21}	78.70	88.06	BLS	29.71
CICIDS2017	99.12	89.19	BLS	1.29

REFERENCES

[1] RIPE NCC [Online]. Available: https://www.ripe.net/analyse/. Accessed: Mar. 20, 2019.

[2] NSL-KDD Data Set [Online]. Available: https://www.unb.ca/cic/datasets/nsl.html. Accessed: Mar. 20, 2019.

[3] Intrusion Detection Evaluation Dataset (CICIDS2017) [Online]. Available: https://www.unb.ca/cic/datasets/ids-2017.html. Accessed: Mar. 20, 2019.

[4] CICFlowMeter [Online]. Available: http://netflowmeter.ca/netflowmeter.html/. Accessed: Mar. 20, 2019.

[5] MS SQL Slammer/Sapphire worm, SANS Institute GIAC Certifications [Online]. Available: https://www.giac.org/paper/gsec/3091/ms-sql-slammer-sapphire-worm/105136. Accessed: Mar. 20, 2019.

[6] Responding to the Nimda worm: recommendations for addressing blended threats, Symantec, Cupertino, CA, USA [Online]. Available: http://securityresponse.symantec.com/avcenter/reference/nimda.final.pdf. Accessed: Mar. 20, 2019.

[7] The Code Red worm, SANS Institute Information Security Reading Room [Online]. Available: https://www.sans.org/reading-room/whitepapers/malicious/code-red-worm-85. Accessed: Mar. 20, 2019.

[8] MathWorks documentation: statistics and machine learning toolbox [Online]. Available: https://www.mathworks.com/help/stats/ Accessed: Mar. 20, 2019.

[9] Broadlearning [Online]. Available: http://www.broadlearning.ai/. Accessed: Mar. 20, 2019.

[10] B. Al-Musawi, P. Branch, and G. Armitage, "BGP anomaly detection techniques: a survey," *IEEE Commun. Surv. Tut.*, vol. 19, no. 1, pp. 377–396, 2017.

[11] P. Batta, Z. Li, and Lj. Trajković, "Evaluation of support vector machine kernels for detecting network anomalies," in *Proc. IEEE Int. Symp. Circuits Syst.*, Florence, Italy, May 2018, pp. 1–4.

[12] A. Ben-Hur and J. Weston, "A user's guide to support vector machines," in *Data Mining Techniques for the Life Sciences, Methods in Molecular Biology*, O. Carugo and F. Eisenhaber, Eds., New York: Springer, 2016, vol. 1415, pp. 223–239.

[13] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: a survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, July 2009.

[14] C. L. P. Chen, Z. Liu, and S. Feng, "Universal approximation capability of broad learning system and its structural variations," *IEEE Trans. Neural Netw. Learn. Syst.*, pp. 1–14, Sept. 2018.

[15] C. L. P. Chen and Z. Liu, "Broad learning system: an effective and efficient incremental learning system without the need for deep architecture," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 1, pp. 10–24, Jan. 2018.

[16] K. Cho, B. van Merriënboer, C. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translations," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, Doha, Qatar, Oct. 2014, pp. 1724–1734.

[17] C. Cortes and V. Vapnik, "Support-vector networks," *J. of Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sept. 1995.

[18] Q. Ding, Z. Li, P. Batta, and Lj. Trajković, "Detecting BGP anomalies using machine learning techniques," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Budapest, Hungary, Oct. 2016, pp. 3352–3355.

[19] Q. Ding, Z. Li, S. Haeri, and Lj. Trajković, "Application of machine learning techniques to detecting anomalies in communication networks," pp. 47–70 and pp. 71–92, in *Cyber Threat Intelligence*, M. Conti, A. Dehghantanha, and T. Dargahi, Eds., Berlin: Springer, 2018.

[20] D. Dolev, S. Jamin, O. Mokryn, and Y. Shavitt, "Internet resiliency to attacks and failures under BGP policy routing," *Comput. Netw.*, vol. 50, no. 16, pp. 3183–3196, Nov. 2006.

[21] H. Gao, Y. Peng, Z. Dai, and H. Li, "Techniques and research trends of network testbed," in *Proc. 10th Int. Conf. Intell. Inf. Hiding Multimedia Signal Process.*, Kitakyushu, Japan, Dec. 2014, pp. 537–541.

[22] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: techniques, systems and challenges," *Comput. Secur.*, vol. 28, pp. 18–28, Feb.-Mar. 2009.

[23] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "LSTM: a search space odyssey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017.

[24] Z. Li, P. Batta, and Lj. Trajković, "Comparison of machine learning algorithms for detection of network intrusions," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Miyazaki, Japan, Oct. 2018, pp. 4248-4253.

[25] Z. Li, A. L. Gonzalez Rios, G. Xu, and Lj. Trajković, "Machine learning techniques for classifying network anomalies and intrusions," in *Proc. IEEE Int. Symp. Circuits Syst.*, Sapporo, Japan, May 2019, to appear.

[26] M. C. Libicki, L. Ablon, and T. Webb, *The Defenders Dilemma: Charting a Course Toward Cybersecurity*, Santa Monica, CA, USA: RAND Corporation, 2015.

[27] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissman, "Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation," in *Proc. DARPA Inform. Survivability Conf. Expo. (DISCEX' 00)*, Hilton Head, SC, USA, Jan. 2000, pp. 12–26.

[28] Z. Liu and C. L. P. Chen, "Broad Learning System: structural extensions on single-layer and multi-layer neural networks," in *Proc. Int. Conf. Secur., Pattern Anal., Cybern.*, Shenzhen, China, Dec. 2017, pp. 136–141.

[29] A. Lutu, M. Bagnulo, C. Pelsser, O. Maennel, and J. Cid-Sueiro, "The BGP visibility toolkit: detecting anomalous Internet routing behavior," *IEEE/ACM Trans. Netw.*, vol. 24, no. 2, pp. 1237–1250, Apr. 2016.

[30] I. Sharafaldin, A. Gharib, A. H. Lashkari, and A. A. Ghorbani, "Towards a reliable intrusion detection benchmark dataset," *Softw. Netw.*, vol. 2017, no. 1, pp. 177–200, Jul. 2017.

[31] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inform. Syst. Secur. Privacy (ICISSP)*, Funchal, Portugal, Jan. 2018, pp. 108–116.

[32] Y. Song, A. Venkataramani, and L Gao, "Identifying and addressing reachability and policy attacks in 'secure' BGP," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2969–2982, Oct. 2016.

[33] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. IEEE Symp. Comput. Intell. Secur. Defense Appl.*, Ottawa, ON, Canada, July 2009, pp. 1–6.

[34] V. Vural and J. G. Dy, "A hierarchical method for multi-class support vector machines," in *Proc. Int. Conf. Mach. Learn., ICML 2004*, Banff, AB, Canada, July 2004, pp. 831–838.