SuperB: Superior Behavior-based Anomaly Detection Defining Authorized Users' Traffic Patterns

Daniel Y. Karasek
dept. of Computer Science
Kennesaw State University
Marietta, Georgia, USA
dkarasek@students.kennesaw.edu

Jeehyeong Kim
dept. of Computer Science and Engineering
Hanyang University
Ansan, South Korea
manje111@hanyang.ac.kr

Victor Youdom Kemmoe
dept. of Computer Science
Kennesaw State University
Marietta, Georgia, USA
vyoudomk@students.kennesaw.edu

Md Zakirul Alam Bhuiyan
dept. of Computer and Information Sciences
Fordham University
Bronx, New York, USA
m.bhuiyan.dr@ieee.com

Sunghyun Cho
dept. of Computer Science and Engineering
Hanyang University
Ansan, South Korea
chopro@hanyang.ac.kr

Junggab Son
dept. of Computer Science
Kennesaw State University
Marietta, Georgia, USA
json@kennesaw.edu

***Abstract*—Network anomalies are correlated to activities that deviate from regular behavior patterns in a network, and they are undetectable until their actions are defined as malicious. Current work in network anomaly detection includes network-based and host-based intrusion detection systems. However, most of them suffer from high false detection rates due to the base rate fallacy. To overcome such a drawback, this paper proposes a superior behavior-based anomaly detection system (SuperB) that defines legitimate network behaviors of authorized users in order to identify unauthorized accesses. We define the network behaviors of the authorized users by training the proposed deep learning model with time-series data extracted from network packets of each of the users. Then, the trained model is used to classify all other behaviors (we define these as anomalies) from the defined legitimate behaviors. As a result, SuperB effectively detects all anomalies of network behaviors. Our simulation results show that the proposed algorithm needs at least five end-to-end conversations to achieve over 95% accuracy and over 93% recall rate. Some simulations show 100% accuracy and recall rate. Our simulations use live network data combined with the CICIDS2017 data set. The performance has an average of less than 1.1% false-positive rate with some simulations showing 0%. The execution time to process each conversation is 85.20 $\pm$ 0.60 milliseconds (ms), and thus it takes about only 426 ms to process five conversations to identify anomaly.**

***Index Terms*—Anomaly Detection, Network Anomaly, Deep Learning, Classification, Behavior identification**

I. Introduction

Cyber attacks on computer networks are becoming a huge problem as network connectivity and data increases [1], [2], [3]. When an attacker gains access to a network, internal network security mechanisms require some type of identification of activities to detect the attacker. Attackers have learned to masquerade as legitimate users or avoid certain actions to avoid detection [1]. However, the behavior of the attackers differs to some degree from that of normal users. There are established patterns of behavior within network data that are unique to each network. Anomaly detection takes advantage of these patterns to identify abnormal behaviors within a network by creating a general baseline of normal behavior. This anomaly detection approach can be greatly beneficial to networks with a group of consistent users, such as a private office or a research laboratory network.

Anomaly detection is a large focus of network-based intrusion detection systems (NIDS), which is a classification of intrusion detection systems (IDS) [3], [4]. Many NIDS using a supervised learning approach have high false-positive rates due to the constantly changing behavior and the diversity of networks [2]. Even, they require a challenging task of obtaining attack-free network data which will be used to train the supervised learning. Unsupervised and semi-supervised approaches can be effective for anomaly detection as they can contribute to reducing false-positive rates by the capability of detecting unknown anomalies [5], [6]. Semi-supervised and hybrid approaches comprise the majority of work in anomaly-focused NIDS and show superiority to supervised learning approaches [2], [7]. Host-based intrusion detection systems (HIDS) are another classification of IDS that only focus on particular host systems. They are similar to NIDS as they focus on anomaly detection and can suffer from high false positives without hybrid approaches [8].

Minimizing false positives is crucial as most anomaly detection schemes have the Bayesian base rate fallacy, which states that near-zero false positives are required for low false detection rates [9]. The false-positive rate must be less than

This research was supported by the MSIT (Ministry of Science, ICT), Korea, under the High-Potential Individuals Global Training Program (2019-0-01601) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation)

the observed event detection rate which for anomalies on a network is low compared to normal traffic. Even with 100% recall and low false positives, the false-detection rate can still be high. Despite there being much research in machine learning techniques for anomaly detection, not much machine learning is implemented in live networks due to solutions not being industrially viable and requiring supplemental detectors to handle false anomaly detections [1], [2], [9], [10]. Additionally, there has not been much exploration for a parameter for anomaly detection with machine learning solutions that corresponds with decreasing false positives and false detections.

In this paper, we propose a superior behavior-based anomaly detection system (SuperB), which defines authorized users' legitimate network behaviors so as to classify all other behaviors against them. We treat the other behaviors as anomalies. To the best of our knowledge, SuperB is the first anomaly detection scheme that defines authorized users' network behavior to detect network anomalies for standard network traffic. Our scheme can be applied to any network traffic with proper formatting of the input network data.

For the effective demonstrations of SuperB, we perform comprehensive simulations with two datasets: (a) network conversation data from the CICIDS2017 dataset [11] and (b) network conversation data collected from individuals in our laboratory. All data is formatted with the open-source Wireshark software, facilitating industry adoption of SuperB. In our simulations, we achieved over 97% accuracy and a false-positive rate of below 1.2%. These results are obtained only using five to eight days of user data with one day left out of learning and used for testing our model. These results are from using a combination of both data sets listed above and from using only our lab data. This is discussed further in the Results section.

II. Related Work

In statistical anomaly detection research, a study proposed a Principal Component Analysis (PCA) based anomaly detection scheme for high-dimensional network flow [12]. Later, the robust statistics-based approach utilizes fitting and flagging on the network data for an enhancement to the existing PCA detection methods [13].

Classification-based schemes are prominent research trends in anomaly detection. In the studies of utilizing Support Vector Machine (SVM), several attempts have been made to improve the classification performance by combining various schemes with conventional SVM-based detection models. Authors in [14] apply One-Click Neural Network (OC-NN) that utilizes One-Click SVM (OC-SVM) as a loss function. Employing Deep Belief Network (DBN) also shows meaningful result in generic feature extraction for anomaly detection, as demonstrated in [15]. According to [4], applying Weighted Chi-Square (WCS) is another option for improving detection rates through discretization process that reduces the dimensionality of data. In [16], a restricted Boltzmann machine (RBM) is proposed for network anomaly detection in a semi-supervised learning approach that trains on normal traffic data only for a more general definition of abnormal behavior.

Long Short-Term Memory (LSTM) architecture is another prominent method for anomaly detection. The study in [17] generates an LSTM-based language model to predict communications between IPs and analyze prediction error to detect traffic outliers. According to [18], detection performance can be improved through the combination of Convolutional Neural Network (CNN), LSTM, and Deep Neural Network (DNN) for complex feature extraction. As noted in [19], DNN-LSTM log pattern detection model that automatically detects anomalies through the analysis of deviated log data also can be used. Authors in [8] employ Gated Recurrent Unit (GRU) and CNN to provide a better detection rate than the traditional LSTM method.

III. Problem Definition

Network behavior is defined as sequential bidirectional flow-based data among various sources and destinations. Normal behavior within a network can be defined as the most common behaviors exhibited over a period of time. Normal behavior must be of non-malicious intent with regards to an assumed purpose of a network. Abnormal behavior can be regarded as a complement of normal behavior. A more refined definition is any uncommon behavior of malicious intent that diverges from an assumed purpose of a network. Normal and abnormal behavior cover the complete set of behaviors that can be exhibited within a network.

In traditional approaches without machine learning, anomaly detection relies on normal network behavior that has been defined by the complement of known abnormal behavior or a broad definition of normal behavior. When a broad definition of normal behavior is used, each network's normal behavior will not be fully defined due to network diversity, so only the complement definition is examined. Consider a group G of all behaviors in an arbitrary network. Let each $S_i \in G$ be a different set of behaviors within the group of behaviors G. Let S_n be all defined normal network behavior, S_m be all unknown behavior, and $G \setminus (S_n \cup S_m)$ be all known abnormal behavior. We have the following as the traditional definition of normal behavior

$$\overline{G \setminus (S_n \cup S_m)} \equiv (S_n \cup S_m)$$

and the following assumption that

$$|(S_m \setminus (S_n \cap S_m)) \setminus ((G \setminus (S_n \cup S_m)) \cap S_m)| > 0$$

This assumption states that there exists some unknown abnormal behavior within S_m. Assuming otherwise claims that a network security system recognizes all possible vulnerabilities. We see that $(S_n \cup S_m) \not\equiv S_n$ meaning this definition of normal network behavior includes some unknown abnormal behavior.

Creating a general baseline of normal network behavior has not been adopted uniformly due to the diversity in how networks are structured and function [2]. However, users are a commonality among networks, and their behavior within a

TABLE I: Selected Features From Conversations

Col	Name	Description
0	InternalExternalAddA	Internal/External Address A
1	AddA	User Address A
2, 3	InternalExternalAddB	Internal/External Address B
4, 6	InternalExternalPortB	Internal/External/Reserved Port B
7	Packets	Total Packets
8	AvgBytesPerPacket	Average Bytes Per Packet
9	PacketsAtoB	Packets From Address A to B
10	AvgBytesPerPacketAtoB	Average Bytes Per Packet From Address A To B
11	PacketsBtoA	Packets From Address B to A
12	AvgBytesPerPacketBtoA	Average Bytes Per Packet From Address B To A
13	Durations	Duration
14, 17	AddressesPreviously	Address A and B Previously Listed
18	KbPerSecAtoB	Kb Per Second From Address A To B
19	KbPerSecBtoA	Kb Per Second From Address B To A
20, 98	AddB	Top Seven Address B From Each User's Training Data With Each Column Mapped To An Address

network can be classified through packet data analysis. Defining the normal behavior of individual users in a network can serve the same purpose of defining normal network behavior. Combining each user's behavior in a binary classification of 'user' or 'others' creates a formalized definition of normal behavior that can be used to better classify known and unknown abnormal behavior as anomalous.

IV. Proposed Scheme

A. Structure of Data

The traffic data format is a subset of features found in the conversation view of Wireshark. This can be seen in Table I. The subset of features includes both nominal and cardinal data. The nominal data is translated into vector labels with each entry in the vector as a different named type of data. For example, a binary vector of length two is denoted (1,0) for an internal IP address for Address B, while a vector of (0,1) is used for an external IP address. Address B port numbers are divided into a length of three binary vector that separates out well-known ports 0 to 1023, registered ports 1024 to 49151, and private ports 49152 to 65535. Addresses previously visited by each user are divided into a length four binary vector to capture all possible combinations where either Address A or Address B is visited previously. Address B is broken into a k-length vector where $k-1$ is a quantity of top addresses visited by users. The quantity of top addresses is 79 for the different traffic data used for our research to serve as padding to reach a total column count of 98 columns to match parameters in our model. One entry in the vector for Address B is a catchall case for any address not found within the top $k-1$ addresses of Address B. These nominal data vectors comprise the columns within the input data format for our model. The subset of features exclude the distinct source IP addresses, labeled as Address A in Wireshark and AddA in Table I, but include a single true or false label to ensure the source address is from a device on the network. Similarly, a single true or false label is used for classifying Address A as an internal or external IP address. Ports used by Address A are also excluded. These modified and excluded features do not contribute much insight into the behavior of each user. The

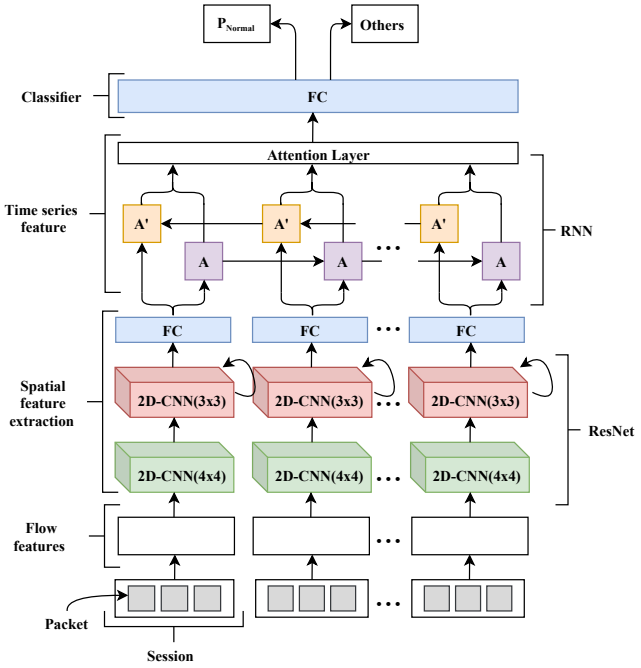


Fig. 1: Proposed Neural Network Architecture

cardinal data is z-score normalized with further outlier removal to remove bias and provide cleaner data for the model. Each of the cardinal data is a single column within the data format and is clearly distinguishable in Table I.

B. Model

Our proposed model for SuperB is a deep learning network comprised of a nine-layer residual neural network (ResNet) with an added fully connected layer that feeds into two bidirectional recurrent neural network (RNN) with an added attention layer. The ResNet is for spatial pattern detection to find relations between different columns in the network data that is used in our model. The ResNet can detect behavior patterns a human would not understand. The RNN is for time-series pattern detection to find complex relations among

different time-series data that humans would not understand. For instance, there might be a pattern between time-series data separated by three time-series that the RNN can detect. Following the RNN is another fully connected layer. In total, there are 16 layers in our model. Figure 1 shows the general architecture of the deep learning network. The first two layers deal with data collection and transformation. The first layer is the data input of the labeled data. In Figure 1, normal behavior is labeled as P_1 and can be any desired user endpoint in a network that has consistent behavior patterns. The second layer is the flow features layer, which is the process of feature selection and formatting of the input data. The feature selection is the subset of the features found within the conversation filtering of Wireshark. The formatting translates any nominal features into a binary output where selected nominal features are made probabilistic and readable by the deep learning network. The formatting of the data is discussed in more depth previously.

The next 10 layers consist of the ResNet and a fully connected layer, which follows the standard convolutional neural network (CNN) architecture for ResNets. Each layer in the ResNet has 64 filters. The first layer of the ResNet has filters of size four by four. The remaining eight layers have filters of size three by three and comprise the main section of the ResNet in pairs of layers with a skip connection after each pair. When data is fed into the ResNet, random samplings of the data are taken in some batch size b with a time-series value t. So each load of data for the ResNet has $b * t$ conversations. The conversations are each resized to dimensions of the number of features. For our experiments, the dimensions are 14 by 7 with the number of total features at 98. The filters in the ResNet layers compress the data and discover hidden patterns among each conversation. The following layer is a fully connected layer where the data is flattened and reshaped into dimensions of b by t by n where n is the number of embedded layers that store spatial relationship data for each t. These 10 layers perform spatial feature extraction on the data to learn feature patterns of each conversation.

The next three layers are the RNN. In the first two layers of the RNN, long short term memory (LSTM) cells are used and are represented by A and A'. The LSTM cells establish patterns from each grouping of t by looking at the sequential data. The third layer is an attention layer that refines the RNN output by using all previous information learned about the inputs from the fully connected layer of the ResNet. The attention layer connects to all of the hidden states for each input. The attention layer creates weights on more relevant input features so that more attention is given to them while extracting patterns from the data. These three layers look for patterns among the time-series of conversations from each person.

The last layer in the architecture is another fully connected layer that uses the spatial feature extraction from the ResNet and the time-series extraction from the RNN to classify each person's data as either normal or abnormal. In Figure 1, this classification is split into P_{Normal} and Others, as P_{Normal} is the selected normal behavior used in our testing.

Algorithm 1: Training the Deep Learning Model

```
Result: Classification Accuracy of Testing Data
Function classify(data_1, data_2) is
    data_normal, data_else = process(data_1, data_2);
    # data_normal_label is [1,0], normal class
    # data_else_label is [0,1], complement class
    return data_normal, data_else;
end
while do_training do
    load(training_data, testing_data);
    if results then
        input = results + training_data;
    else
        input = training_data;
    end
    labeled_input = training_labels(input);
    trained = train(labeled_input);
    tested = classify(training_data, testing_data);
    tr_Cost = trained.get_tr_Cost();
    if tr_Cost ~ 0 then
        do_training = FALSE;
        return trained, tested;
    else
        results = trained;
        print trained, tested;
    end
end
```

C. Training

Algorithm 1 shows the general layout of the training for our model. Each step in the training outputs training and testing accuracy. In Algorithm 1, the classify function is detailed to show the output labels. The labels outputted by the classification function are discussed in the next section. The other functions used are self-explanatory. The algorithm loads data and does training and testing on the loaded data. The loaded data for training is either four or seven days of user data, and the testing data is one day. This is later discussed in the Results section. The train function has a tr_Cost value, which is the difference measure between the input labels and the output labels from the deep learning model. The tr_Cost eventually converges to zero after an adequate number of training steps. When the tr_Cost is approximately zero, training of the model is finished, and the final trained and tested results are outputted. Otherwise, the trained and tested results for that step are outputted, and the training data is fed back into the model for the next training step. The results include the training step number as well as the accuracy of training and testing data.

The input data from each person is separated with at least one day being test data denoted P_ite and the remaining days being training data denoted P_itr where i represents a person. Here the test data is not used for learning. One person i is labeled normal behavior. All others are either labeled abnormal

behavior or unlabeled abnormal behavior. There is a testing set of people that is a complement to the learning set, which consists of unlabeled abnormal behavior. This testing set of people is not used for learning. The learning set must have at least three people and at most $|G \setminus (S_n \cup S_m)|$ - 1 as to allow for generalization of abnormal behavior and still have at least one unlabeled test person.

To avoid overfitting, we implement early stopping while training our model. This is how we ended up with using 10,000 steps for training SuperB. We initially train with 20,000 steps capturing intervals of every 100 training steps. After many tests we determined that 10,000 steps was sufficient.

D. Classifying Behavior

Our algorithm is tailored for recognizing a specific person's behavior as baseline normal while all other people's behavior is considered abnormal. As described in Figure 1 and Algorithm 1, the output of our model classifies people's behavior as either normal, in our case a specific person, or abnormal, in our case all other people. The output labeled as other people differs slightly from its input counterpart. The input counterpart is labeled abnormal behavior. However, in the last classification part of our algorithm, it intentionally makes the output class of all abnormal behavior, both labeled and unlabeled, a complement class with the same label as the input abnormal behavior. In semi-supervised learning, deep learning networks can create a new default class as a catch all for unknown output data. Using this complement class can better isolates the normal behavior features from all of the unknown abnormal behavior features, which helps classify unlabeled traffic correctly. This concept allows for some tailoring of known abnormal behavior detection while addressing unknown abnormal behavior. However, this default class concept is a two way street. A default class can be created to classify unlabeled behavior as normal if it better fits the general pattern of labeled normal behavior. This was discovered in some of our initial experiments. It is recommended to use more labeled abnormal behavior data than labeled normal behavior data so the abnormal default class catches unlabeled abnormal behavior.

Our current classification scheme can be applied to two scenarios, a single user and a network of users. The single user case follows how the classification scheme is written. For the network of users case, each user within the network would require a model trained on their normal behavior. Once all of the models are trained, the models can be used in parallel to check against new traffic in the network. If the behavior does not align with any of the trained models, said behavior can be labeled as abnormal and proper protocols can be taken.

V. Simulations

A. Gathering Data

The data used in testing our algorithm is from two sources. The shortened labels for the data can be found in Table II. The first source is the publicly available CICIDS2017 data set. As stated in [11], the CICIDS2017 data set was developed with realistic background traffic using the B-Profile system designed in [20]. This data was captured using port mirroring, so it is a complete capture. From CICIDS2017, data from seven of the machines are used, including the Windows 10 Pro 32b, Windows 10 64b, Ubuntu 14.4 32b, Ubuntu 14.4 64b, Ubuntu 16.4 32b, Ubuntu 16.4 64b, and MAC. These machines are considered to be users for the sake of our experiments. All five days of the CICIDS2017 data set are used for our experiment with only four days used in the training set and the remaining day used for testing. In some preliminary testing, the Windows 7 Pro from CICIDS2017 showed very strange behavior, so it has been left out of testing. In simulations using the Windows 7 Pro as labeled normal or labeled abnormal behavior, our model would not learn correctly. Additionally, using the Windows 7 Pro for unlabeled abnormal behavior gave varying results between high and low accuracy. More analysis of the simulated user behaviors in the CICIDS2017 data set is required.

TABLE II: Labels and Sources of Collected Data

Label	Source	Machine
P1	CICIDS2017	Ubu 16.4 64b
P2	CICIDS2017	Ubu 14.4 32b
P3	CICIDS2017	Ubu 16.4 32b
P4	CICIDS2017	Ubu 14.4 64b
P5	CICIDS2017	Win 10 Pro 32b
P6	CICIDS2017	Win 10 64b
P7	CICIDS2017	Mac OS X
P8	OfficeB	Win 10 Enterprise 64b
P9	OfficeB	Win 10 Enterprise 64b
P10	OfficeB	Win 10 Enterprise 64b
P11	OfficeB	Win 10 Enterprise 64b
P12	OfficeA	Win 10 Enterprise 64b
P13	OfficeA	Win 10 Enterprise 64b
P14	OfficeA	Win 10 Enterprise 64b
P15	OfficeA	Win 10 Enterprise 64b
P16	OfficeA	Win 10 Enterprise 64b

The second source is data collected from two network groups within the Information and Intelligent Security (IIS) Laboratory at Kennesaw State University. All computers used in this data source are Windows machines running Windows 10 64B. These groups are named OfficeA and OfficeB based on their respective room numbers at the college. The OfficeA data group consists of five users and was collected between September 17th, 2019, and September 30th, 2019. The OfficeB data group consists of four users and was collected between January 29th, 2020, and February 13th, 2020. The collection period for all IIS network data was between 9 AM and 4 PM for at least five hours per collection day. Both the OfficeA and OfficeB data groups consist of eight days of collected network data per user. The network data in the ISS data set was collected using port mirroring to allow for a complete capture. When using the IIS data set with the CICIDS2017 data set, only five out of the eight days are used with four days used for training and the remaining day used for testing.

For both the CICIDS2017 and IIS data set, multiple days of data are used to fully cover the behavior pattern of users within a network. It is assumed that a user's behavior will

TABLE III: Arrangements Of Data For Simulations

Arrangement	Labeled Normal	Labeled Abnormal	Unlabeled Abnormal
A1	OfficeB (P8-P11)	CICIDS2017 (P2, P5, P7)	CICIDS2017 (1 from P3, P4, P6)
A2	OfficeB (P8-P11)	CICIDS2017 (P1-P3)	CICIDS2017 (1 from P4, P6, P7)
A3	CICIDS2017 (P1, P5-P7)	OfficeB (P8-P10)	OfficeB (P11)
A4	OfficeA (P12-P15)	CICIDS2017 (P1-P3)	CICIDS2017 (1 from P4, P6, P7)
A5	CICIDS2017 (P1, P2, P5, P7)	OfficeA (P12-P14)	OfficeA (1 from P15-P16)
A6a	OfficeA (P12)	OfficeA (3 from P13-P16)	OfficeA (1 from P13-P16)
A6b	OfficeA (P13)	OfficeA (3 from P12, P14-P16)	OfficeA (1 from P12, P14-P16)
A6c	OfficeA (P14)	OfficeA (3 from P12, P13, P15, P16)	OfficeA (1 from P12, P13, P15, P16)
A6d	OfficeA (P15)	OfficeA (3 from P12-P14, P16)	OfficeA (1 from P12-P14, P16)
A6e	OfficeA (P16)	OfficeA (3 from P12-P15)	OfficeA (1 from P12-P15)
A7	OfficeA (P12)	OfficeB (3 from P8-P11) & CICIDS2017 (P1, P5, P7)	OfficeB (1 from P8-P11) & CICIDS2017 (P2) & OfficeA (P15)
A8	OfficeA (P12)	OfficeB (3 from P8-P11) & CICIDS2017 (P1, P3, P4)	OfficeB (1 from P8-P11) & CICIDS2017 (P2) & OfficeA (P15)

remain relatively the same when observed a day at a time. However, some behavior activity occurs in cycles with days between the activity. Some examples include paying a monthly bill and online ordering of goods. Thus, at least a full business week or five days of data collection is highly suggested for our model.

For both the CICIDS2017 and IIS data sets, Wireshark is used to read the PCAP data. For the IIS data set, Wireshark is used to capture the PCAP data as well. Within Wireshark, the conversation view for each PCAP is used to extract a CSV file with metadata on the activity within the network during the time frame of the capture. The CSV files are converted into a machine learning readable format that focuses on the behavior of each user. The features collected and their format are previously discussed in Section IV under Structure of Data.

B. Environmental Settings

For our model, we test ten different arrangements using combinations of the CICIDS2017 and the dataset collected from our laboratory (named IIS data sets). In total there are 32 simulations tested using these arrangements. This is to ensure that multiple scenarios are tested. The arrangements are designated as A1 to A6 and are based around what sources are used as labeled and unlabeled behavior. A6 has 5 sub-arrangements.These can be found in Table III.

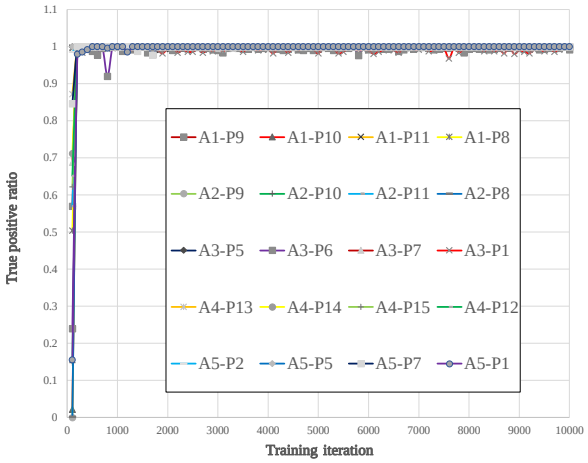
All arrangements except A6, A7, and A8 consist of four model training sessions with each model trained on a different normal behavior pattern and three abnormal behavior patterns. Each model has unlabeled behavior tested against each training step. Arrangement A6 only uses OfficeA and consists of five sub-arrangements of four model training sessions with each sub-arrangement training on the same normal behavior but with different configurations of labeled and unlabeled abnormal behavior. Each of these five sub-arrangements is averaged to find the individual accuracy of all five users within OfficeA. Arrangement A6 is done this way to better illustrate the strength of our algorithm and model against live network data. A7 and A8 have sub-arrangements based around using different groups of users from OfficeB for training and testing. This is discussed more in the Results section. For testing labeled and unlabeled behavior from arrangements A1 to A5, data from Tuesday is used from CICIDS2017, and data from the fifth collection day is used from OfficeB and OfficeA. These days are arbitrarily selected; however, Monday for CICIDS2017 should be only be used for labeled behavior due to it containing all benign behavior. For arrangement A6, the sixth day was used for testing.

TABLE IV: Average Test Performance for A1-A5 with 10K Training Steps

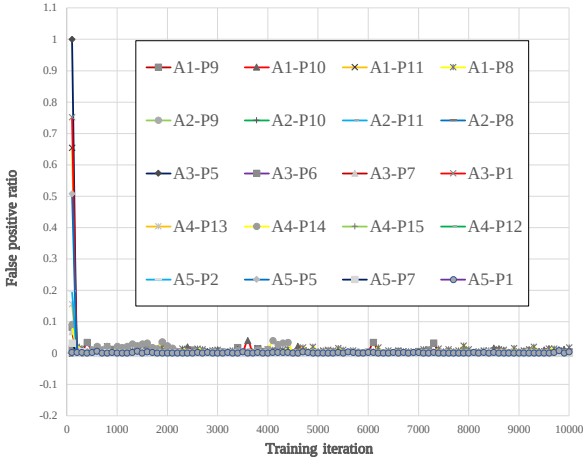
Test Data	TPR	FPR	Acc	Prec	F1
A1-P8	1	0	1	1	1
A1-P9	1	0	1	1	1
A1-P10	1	0	1	1	1
A1-P11	1	0.008	0.996	0.992	0.996
A2-P8	1	0	1	1	1
A2-P9	1	0	1	1	1
A2-P10	1	0	1	1	1
A2-P11	1	0	1	1	1
A3-P1	1	0	1	1	1
A3-P5	0.996	0	0.998	1	0.998
A3-P6	1	0	1	1	1
A3-P7	0.990	0.004	0.993	0.996	0.993
A4-P12	1	0.002	0.999	0.998	0.999
A4-P13	1	0.002	0.999	0.998	0.999
A4-P14	1	0.002	0.999	0.998	0.999
A4-P15	1	0	1	1	1
A5-P1	1	0	1	1	1
A5-P2	1	0	1	1	1
A5-P5	1	0	1	1	1
A5-P7	1	0	1	1	1

Test Data = Arrangement-Labeled Normal, **TPR** = True Positve Rate, **FPR** = false-positive Rate, **Acc** = Accuracy, **Prec** = Precision, **F1** = F-Measure.

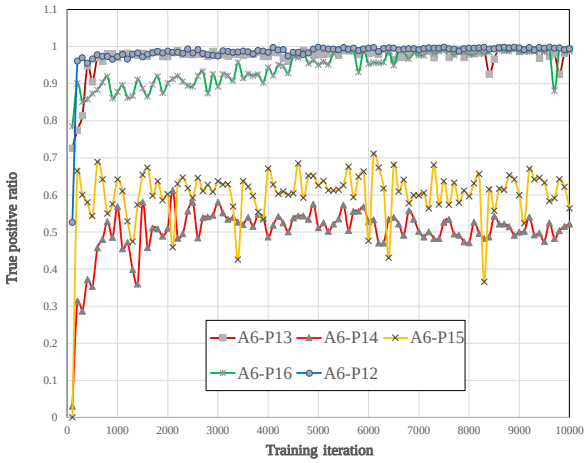
In A1, OfficeB is normal behavior, and CICIDS2017 is abnormal behavior. A1 has labeled abnormal behavior that consists of Windows, Ubuntu, and Mac users with unlabeled abnormal behavior consisting of Linux and Windows users. In A2, OfficeB is normal behavior, and CICIDS2017 is abnormal behavior. A2 has labeled abnormal behavior as only Ubuntu users and unlabeled abnormal behavior as Windows, Ubuntu, and Mac users. This difference between A1 and A2 is due to some preliminary testing that showed behavior patterns were similar among users with shared operating systems in the CICIDS2017 data set. More analysis of the CICIDS2017



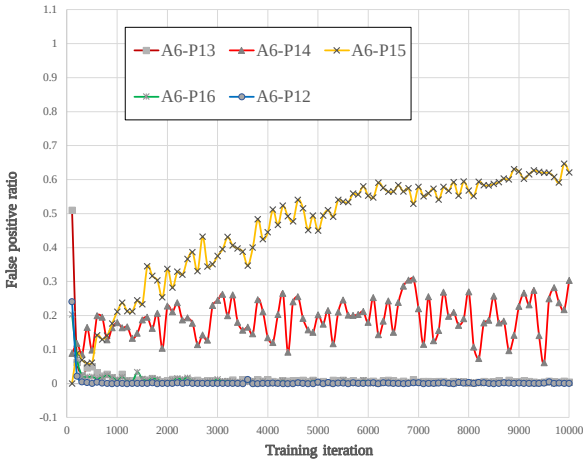
(a) True Positive Ratio for Arrangements A1 to A5



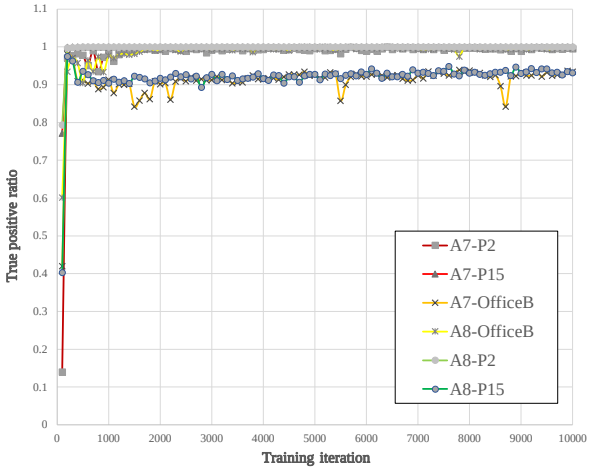
(b) false-positive Ratio for Arrangements A1 to A5



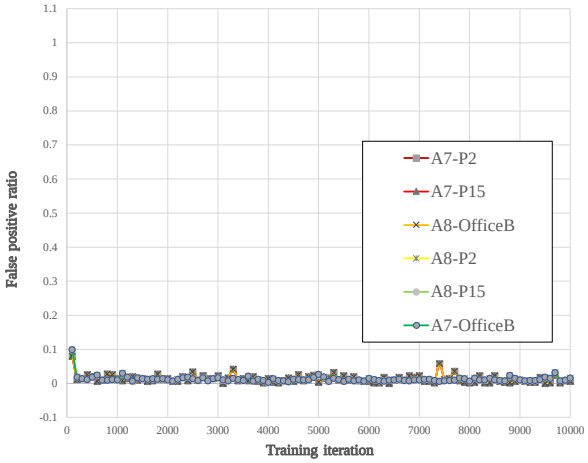
(c) True Positive Ratio for Sub-arrangements A6



(d) false-positive Ratio for Sub-arrangements A6



(e) True Positive Ratio for Sub-arrangements A7 and A8



(f) false-positive Ratio for Sub-arrangements A7 and A8

Fig. 2: Experimental Results based on CICIDS2017 and IIS datasets from using batch sizes of 500. These results come from the testing set of data for each user. True positives are for correct labeling of abnormal behavior. false positives are for mislabeled normal behavior as abnormal.

TABLE V: Average Test Performance for A6 with 10K Training Steps

Test Data	TPR	FPR	Acc	Prec	F1
A6a	0.988	0.006	0.991	0.995	0.991
A6b	0.512	0.155	0.678	0.598	0.531
A6c	0.599	0.650	0.475	0.597	0.476
A6d	0.970	0.002	0.984	0.999	0.983
A6e	0.998	0.002	0.994	0.998	0.994

TPR = True Positve Rate, **FPR** = false-positive Rate, **Acc** = Accuracy, **Prec** = Precision, **F1** = F-Measure.

TABLE VI: Average Test Performance for A7 & A8 with 10K Training Steps

Test Data	TPR	FPR	Acc	Prec	F1
A7-P12-(P8-P11)	0.997	0.002	0.9975	0.997998	0.997499
A7-P12-P2	1	0.002	0.999	0.998004	0.999001
A7-P12-P15	0.93	0.002	0.964	0.997854	0.962733
A8-P12-(P8-P11)	0.9935	0.011	0.99125	0.989049	0.99128
A8-P12-P2	1	0.011	0.9945	0.98912	0.99453753
A8-P12-P15	0.922	0.011	0.9555	0.98821	0.953958

Test Data = Arrangement-Labeled Normal-Unlabeled Abnormal, **TPR** = True Positve Rate, **FPR** = false-positive Rate, **Acc** = Accuracy, **Prec** = Precision, **F1** = F-Measure.

data set is required to formally define the number of unique behavior patterns used, which is outside the scope of this paper. Using only CICIDS2017 to test our model gives results that could stem from some bias in how the traffic is created, so a combination of CICIDS2017 and the IIS data is used as there is a distinct difference between the two data sets. The bias with CICIDS2017 stems from how our model can both distinguish the different users but also can distinguish the different operating systems. Our model's ability to distinguish the different operating systems may be related to the 25 simulated user behaviors found within the seven simulated users we use from CICIDS2017. From the 25 simulated user behaviors, it is not guaranteed that one behavior is not spread out over multiple machines. Thus, A1 covers the scenario when unlabeled abnormal behavior is possibly identical to labeled abnormal behavior. In A3, CICIDS2017 is normal behavior, and OfficeB is abnormal behavior. As each user in OfficeB has unique behavior, an arbitrary order is selected. The labeled normal behavior from CICIDS2017 has Windows, Ubuntu, and Mac users represented. A4 and A5 are the same as A2 and A3, respectively, except using OfficeA instead of OfficeB. In A6, OfficeA is the only data set used. In A7 and A8, we use a mix of all three data sets. Labeled normal behavior is set to P12 (OfficeA) bot both A7 and A8. In A7 labeled abnormal behavior is set to a Windows, Ubuntu, and Mac machine from CICIDS2017 as well as a selection of three users from OfficeB. In A8, the labeled abnormal behavior is similar except that only Ubuntu machines are used from CICIDS2017. In both A7 and A8, the unlabeled abnormal behavior is one user from OfficeB who was left out of the labeled behavior, P2 from CICIDS2017, and P15 from OfficeA.

The deep learning models are trained with 10,000 steps to ensure the convergence of the cost output from the network and to ensure that all of the input data is selected by our random batch selection for each iteration. We train all 24 different models at using a time-series t rate of 5. This time-series amount is enough for distinguishing behavior patterns and small enough for our models to be applicable in real-time analysis of network traffic. We use a batch size b of 128 for loading training data and an embedded size n of 40 in the ResNet. The testing batch size is 500 for better accuracy.

C. Results

Figure 2 represents the experimental resuts based on two datasets: CICIDS2017 and IIS. Figure 2a, Figure 2c, and Figure 2e depict changes of the true positive ratios (TPR) over the 10,000 training iterations. All converge to a true positive ratio of nearly one. The spikes generated training steps where new data is learned as 10,000 isn't guaranteed to cover all of the network data. More training can give better convergence. The changes in the false-positive ratios (FPR) over 10,000 training steps are seen in Figure 2b, Figure 2d, and Figure 2f. The FPR also has some spikes due to the model learning new data.

The test performance for each arrangement after 10,000 training steps can be found in Table IV, Table V, and Table VI. Table IV has arrangements A1 to A5, Table V has the sub-arrangements of A6, and Table VI has sub-arrangements of A7 and A8. This separation is because there are separate arrangement types with the main difference in the structure of the labeled normal behavior and sources of data. As discussed previously, the TPR converges very close to a ratio of one, and the FPR converges very close to zero. These results maintain an average accuracy above 97% and an execution time of around 100 ms after training the model.

In Table VI, we did a combination of data different from all other arrangements. This combination is done to further show the strength of our model and to remove bias from our testing. In our results found in Table VI, the average of four simulations per sub-arrangement is used. In the selection of both labeled and unlabeled behavior, we picked an arbitrary grouping from the CICIDS2017 and IIS data sets. The column Test Data represent a combination of arrangement-labeled normal-unlabeled abnormal. We made two arrangements, A7 and A8, to show results when using a variety of machines from the CICIDS2017 data set while still using a variety of machines from our own collected data set. Some of the more surprising results come from testing with P15. P15 is from OfficeA, and OfficeA is not used in labeled abnormal behavior. This shows that this truly unknown behavior, as in behavior not from the same network, is labeled as abnormal behavior. Thus, our model is very effective in using live network data.

These results suggest our algorithm is successful in using

behavior to classify unknown user traffic, both when looking at individual users within a network and when comparing user behavior from different networks. The algorithm maintains a minimal FPR less than 1.2% on average, which gives credence to the idea that the time-series pattern for each user is distinct enough for classification. The low FPR means our model is resistant to the base rate fallacy problem. Our algorithm is applicable to real-time traffic analysis. The main constraint is the time it takes for a user to create five conversations, or sessions, worth of traffic. This lag is necessary to collect enough data to establish a pattern of behavior. In some previous testing of our model we found that using larger time-series values increases our model's performance because better patterns of behavior can be identified with each user. However, these time-series values stray more from a real-time solution.

During the experiments, we found some cases that our algorithm does not work well. Figure 2c and Figure 2d, which include A6-P14 (A6c) and A6-P15 (A6d), show these cases. After exploring their data, the behavior of these two users was found to be very similar to each other, which caused learning issues when training our model using either user as labeled normal behavior. Users P14 and P15 at the time were working on the same project collaboratively. To fix these bad results, more data is required to capture the users' normal unique behavior better. Also, having a general work schedule associated with the data collection can help discover strange portions of the data. When discussing our results below, we ignore A6-P14 and A6-P15.

For the efficiency matter, we measured the execution time of the algorithm. Each training step takes 85.20$\pm$ 0.60 milliseconds (ms) with a total training time of approximately 852 second for the 10,000 training steps. Our experiment showed that we achieved high accuracy and low false positives of unknown anomaly detection when using a small time-series of $t = 5$, which execution time is approximately 420 ms. The main technologies used for the testing are an Intel Core i9-7920X CPU, GeForce RTX 2080 Ti GPU, and Tensorflow version 1.12.

VI. Conclusions

In this paper, we proposed a solution for network anomaly detection using a classification approach of individual user behavior. This created a general solution that can be applied to entire networks for anomaly detection network-wide. Our solution is a semi-supervised deep learning approach that showed a high accuracy rate of detecting unlabeled abnormal behavior. To the best of our knowledge, this is the first approach to detect network anomalies by defining legitimate users' normal behaviors using network conversations. The proposed architecture consists of a nine-layer ResNet coupled with a bidirectional RNN that has an attention layer. The results from our experiment showed that we achieved high accuracy and low false positives of unknown anomaly detection when using a small time-series of $t = 5$. The accuracy averaged above 97%, and the FPR remained on average less than 1.2%. The TPR, which is equivalent to recall rate, shows that almost no mislabeling occurred, even when only using live network data. All of these results are from 10,000 training steps on 40 different simulations using our algorithm.

References

[1] C. Gilmore and J. Haydaman, "Anomaly detection and machine learning methods for network intrusion detection: an industrially focused literature review," in *Proceedings of the International Conference on Security and Management (SAM)*, pp. 292–298, 2016.

[2] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. E. Solano, and O. M. C. Rendon, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 16, pp. 1–99, 2018.

[3] M. H. Kamarudin, C. Maple, T. Watson, and N. S. Safa, "A logitboost-based algorithm for detecting known and unknown web attacks," *IEEE Access*, vol. 5, p. 26190–26200, 2017.

[4] W. Mohamed, M. Abdollah, Z. Masud, Y. Robiah, R. Abdullah, and Z. Muda, "Enhance intrusion detection capabilities via weighted chi-square, discretization and svm," *Journal of Theoretical and Applied Information Technology*, vol. 96, pp. 6006–6017, 2018.

[5] N. S. Arunraj, R. Hable, M. Fernandes, K. Leidl, and M. Heigl, "Comparison of supervised, semi-supervised and unsupervised learning methods in network intrusion detection system (nids) application," in *Anwendungen und Konzepte der Wirtschaftsinformatik*, pp. 1–10, 2017.

[6] G. Pang, C. Shen, H. Jin, and A. van den Hengel, "Deep weakly-supervised anomaly detection," in *arXiv:1910.13601v2*, pp. 1–18, 2020.

[7] M. Mazini, B. Shirazi, and I. Mahdavi, "Anomaly network-based intrusion detection system using a reliable hybrid artificial bee colony and adaboost algorithms," *Journal of King Saud University - Computer and Information Sciences*, vol. 31, no. 4, p. 541–553, 2018.

[8] A. Chawla, B. Lee, S. Fallon, and P. Jacob, "Host based intrusion detection system with combined cnn/rnn model," in *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 149–158, Springer, 2018.

[9] S. Axelsson, "The base-rate fallacy and the difficulty of intrusion detection," *ACM Transactions on Information and System Security*, vol. 3.3, pp. 186–205, 2000.

[10] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, (Washington, DC, USA), pp. 305–316, IEEE Computer Society, 2010.

[11] I. Sharafaldin, A. Habibi Lashkari, and A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy*, pp. 108–116, 2018.

[12] M. Ding and H. Tian, "Pca-based network traffic anomaly detection," *Tsinghua Science and Technology*, vol. 21, no. 5, pp. 500–509, 2016.

[13] P. J. Rousseeuw and M. Hubert, "Anomaly detection by robust statistics," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 2, p. e1236, 2018.

[14] R. Chalapathy, A. K. Menon, and S. Chawla, "Anomaly detection using one-class neural networks," *arXiv:1802.06360*, 2018.

[15] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, "High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning," *Pattern Recognition*, vol. 58, pp. 121–134, 2016.

[16] U. Fiore, F. Palmieri, A. Castiglione, and A. Santis, "Network anomaly detection with the restricted boltzmann machine," *Neurocomputing*, vol. 122, pp. 13–23, 2013.

[17] B. J. Radford, L. M. Apolonio, A. J. Trias, and J. A. Simpson, "Network traffic anomaly detection using recurrent neural networks," *arXiv:1803.10769*, 2018.

[18] T.-Y. Kim and S.-B. Cho, "Web traffic anomaly detection using c-lstm neural networks," *Expert Systems with Applications*, vol. 106, pp. 66–76, 2018.

[19] M. Du, F. Li, G. Zheng, and 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*, pp. 1285–1298, ACM, 2017.

[20] I. Sharafaldin, A. Gharib, A. Habibi Lashkari, and A. Ghorbani, "Towards a reliable intrusion detection benchmark dataset," *Software Networking*, vol. 2017, pp. 177–200, 2017.