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

Daniel Karasek

Follow this and additional works at: https://digitalcommons.kennesaw.edu/cs_etd

Part of the Artificial Intelligence and Robotics Commons, and the Information Security Commons

Recommended Citation

This Thesis is brought to you for free and open access by the Department of Computer Science at DigitalCommons@Kennesaw State University. It has been accepted for inclusion in Master of Science in Computer Science Theses by an authorized administrator of DigitalCommons@Kennesaw State University. For more information, please contact digitalcommons@kennesaw.edu.
SuperB: Superior Behavior-based Anomaly Detection
Defining Authorized Users' Traffic Patterns

A Thesis Presented to
The Faculty of the Computer Science Department

by

Daniel Yang Karasek

In Partial Fulfillment
of Requirements for the Degree
Master of Science, Computer Science

Kennesaw State University
May 2020
SuperB: Superior Behavior-based Anomaly Detection
Defining Authorized Users' Traffic Patterns

Approved:

Dr. Junggab Son - Advisor

Dr. Coskun Cetinkaya – Department Chair

Dr. Jon Preston - Dean
In presenting this thesis as a partial fulfillment of the requirements for an advanced degree from Kennesaw State University, I agree that the university library shall make it available for inspection and circulation in accordance with its regulations governing materials of this type. I agree that permission to copy from, or to publish, this thesis may be granted by the professor under whose direction it was written, or, in his absence, by the dean of the appropriate school when such copying or publication is solely for scholarly purposes and does not involve potential financial gain. It is understood that any copying from or publication of, this thesis which involves potential financial gain will not be allowed without written permission.

Daniel Yang Karasek

May 11, 2020
Notice To Borrowers

Unpublished theses deposited in the Library of Kennesaw State University must be used only in accordance with the stipulations prescribed by the author in the preceding statement.

The author of this thesis is:

Daniel Yang Karasek
1100 S Marietta PKWY,
Marietta, GA 30060

The director of this thesis is:

Dr. Junggab Son
1100 S Marietta PKWY,
Marietta, GA 30060

Users of this thesis not regularly enrolled as students at Kennesaw State University are required to attest acceptance of the preceding stipulations by signing below. Libraries borrowing this thesis for the use of their patrons are required to see that each user records here the information requested.
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, network anomaly detection schemes can suffer from high false detection rates due to the base rate fallacy. When the detection rate is less than the false positive rate, which is found in network anomaly detection schemes working with live data, a high false detection rate can occur. 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. I 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 (I define these as anomalies) from the defined legitimate behaviors. As a result, SuperB effectively detects all anomalies of network behaviors. The simulation results show that SuperB needs at least five end-to-end network conversations to achieve over 95% accuracy and over 93% true positive rate. Some simulations achieved 100% accuracy and true positive rate. The 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 ± 0.60 milliseconds (ms), and thus it takes about only 426 ms to process five conversations to identify an anomaly.
ACKNOWLEDGEMENTS

First, I would like to extend a big thank you to my parents for all of the love and support they have given me during my academic career in pursuing my master’s degree. Second, I would like to thank my advisor Dr. Junggab Son for all of the valuable advice and guidance he has given me in completing my research. Third, I would like to extend a thank you to my friends and peers for all of the support they have given me through hearty discussions and words of encouragement. Lastly, I would like to thank both my former dog Max, the OG who is chilling in peace up in heaven, and my current dog Faye for helping me keep my sanity by always being there after endless hours of data collection and processing.
LIST OF TABLES

Table 1: Selected features from conversations.................................................................25
Table 2: Identification of collected data..................................................................................27
Table 3: Arrangements of data............................................................................................29
Table 4: Average performance for A1 - A5 with 10k training steps........................................36
Table 5: Average performance for A6 with 10k training steps..................................................37
Table 6: Average performance for A7 & A8 with 10k training steps........................................37
LIST OF FIGURES

Figure 1: Diagram of a standard convolutional neural network..................................................12
Figure 2: Diagram of residual neural network building block..................................................15
Figure 3: Diagrams of a basic recurrent neural network and a bidirectional
recurrent neural network........................................................................................................17
Figure 4: Diagram of an attention layer architecture................................................................19
Figure 5: Diagram of the SuperB model architecture................................................................24
Figure 6: Algorithm of the deep learning model........................................................................35
Figure 7: False positive ratios for A1 - A5..................................................................................41
Figure 8: False positive ratios for A6.........................................................................................42
Figure 9: False positive ratios for A7 & A8...............................................................................43
Figure 10: True positive ratios for A1 - A5...............................................................................44
Figure 11: True positive ratios for A6.......................................................................................45
Figure 12: True positive ratios for A7 & A8...............................................................................46
# TABLE OF CONTENTS

ABSTRACT.............................................................................................................................iii

ACKNOWLEDGEMENTS...........................................................................................................iv

LIST OF TABLES....................................................................................................................v

LIST OF FIGURES..................................................................................................................vi

1. INTRODUCTION.................................................................................................................1
   1.1 Network Anomaly Detection Introduction.......................................................................1
   1.2 Problem Statement............................................................................................................2
   1.3 Contribution.....................................................................................................................5

2. NETWORK DATA BACKGROUND.......................................................................................6
   2.1 Network Data Format.......................................................................................................6
   2.2 Network Data Analysis Software....................................................................................6

3. RELATED WORKS..............................................................................................................8

4. DEEP LEARNING METHODS............................................................................................11
   4.1 Convolutional Neural Network........................................................................................11
   4.2 Residual Neural Network................................................................................................13
   4.3 Recurrent Neural Network..............................................................................................16
   4.4 Attention Layer Architecture for RNNs..........................................................................18
   4.5 Combining Neural Networks in a Model..........................................................................20

5. SUPERB MODEL................................................................................................................21
   5.1 Model Design..................................................................................................................21
   5.2 Structure of Data and Datasets........................................................................................25
5.3 Experiment Design...........................................................................................................29
5.4 Training..................................................................................................................................33
5.5 Results......................................................................................................................................36
5.6 Future Directions.....................................................................................................................47

6. CONCLUSION.............................................................................................................................48

REFERENCES..................................................................................................................................49
CHAPTER I

INTRODUCTION

1.1 Network Anomaly Detection 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 NIDSs 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]. 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.

1.2 Problem Statement

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, 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 $G\backslash(S_n \cup S_m)$ be all known abnormal behavior. The following is a traditional definition of normal behavior

$$G\backslash(S_n \cup S_m) \equiv (S_n \cup S_m)$$

with the following assumption that

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

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. It is clear that \((S_n \cup S_m) \neq S_n\) 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 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.

The problem addressed in this thesis is the identification of novel abnormal behavior within live networks. The process of identifying this novel abnormal behavior involves establishing a tailored baseline of normal behavior for any particular network based on a general solution. Additionally, recognized attacks can be included in the baseline as known abnormal behavior. Then, any behavior not recognized within the baseline of normal behavior is flagged as either known or novel abnormal behavior. This method must easily work with existing enterprise-level technology as the use case is for live networks at companies.
1.3 Contribution

This thesis proposes a superior behavior-based anomaly detection system (SuperB), which defines authorized users’ legitimate network behaviors so as to classify all other behaviors against them. The system treats behaviors that are not the legitimate behaviors as anomalies. SuperB is a combination of a residual neural network (ResNet) and a recurrent neural network (RNN), which maximizes the detection accuracy while achieving a false-positive rate of near zero percent. The ResNet is composed of convolutional neural networks (CNNs) with many residual blocks. The ResNet extracts behavior found within each user's conversations to find hidden patterns that are unique to each user. The RNN then combines the patterns of multiple conversations discovered by the ResNet to establish a behavior pattern over a short time span. Thus, the model strives to create a true model of a user's normal behavior through a binary classification of normal and abnormal behavior. This binary classification can detect unknown patterns in new data.

To the best of my knowledge, SuperB is the first anomaly detection scheme that defines authorized users' network behavior to detect network anomalies for standard network traffic. This scheme can be applied to any network traffic with proper formatting of the input network data. For the effective demonstrations of SuperB, I perform comprehensive simulations with two datasets: (a) network conversation data from the CICIDS2017 dataset [11] and (b) network conversation data collected from individuals in my laboratory. All data is formatted with the open-source Wireshark software, facilitating industry adoption of SuperB.
CHAPTER II

NETWORK DATA BACKGROUND

2.1 Network Data Format

There is a general format of network data known as a PCAP, which is an abbreviation for packet capture. A PCAP is an application programming interface (API) written in the C programming language that is used in capturing packets sent over a network. A PCAP stores an entire session of network activity, including endpoint-to-endpoint metadata and the data sent between endpoints. The data sent between endpoints is typically endpoint-to-endpoint encrypted and cannot be decrypted solely from looking at a PCAP. A PCAP file can be very large depending on the network being captured, reaching into the tens of gigabytes if not hundreds. The size is determined by how much network traffic occurs on the network while capturing and the duration of the capture.

2.2 Network Data Analysis Software

The most common software used in analyzing PCAP files is Wireshark. This software is open-source and widely used both in education and industry for reading PCAPs. Wireshark is similar to another common software tcpdump that is used for reading PCAP files; however, Wireshark is significantly more user friendly as it has a graphical front-end and integrated filtering of various network features. Wireshark can both read saved PCAPs and capture a new PCAP live from the wire. When reading individual packets in Wireshark, packets that are
encrypted, usually by endpoint-to-endpoint encryption, cannot be decrypted and show up as hidden data masked by some hex code pattern generated by whichever endpoint sent the encrypted packet. When looking at the metadata of packets, Wireshark has many options for different filtering. One such option is called a conversation view. Here conversation is synonymous with session. The conversation view shows all conversations, or sessions, that occur over a network and includes metadata such as IP addresses, ports used, number and speed of packets sent and received, and bits sent over the wire between endpoints.
CHAPTER III

RELATED WORKS

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 results 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 a 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 [20] employ Gated Recurrent Unit (GRU) and CNN to provide a better detection rate than the traditional LSTM Method.

Boosting is an ensemble classification technique enhancing the accuracy of weak classifiers. The two representative boosting algorithms proposed for anomaly detection are Adaboost and Logitboost [20], [21]. The studies employed Adaboost and Logitboost each in anomaly detection researches to gain a high detection rate and low false-positive rate.

In the GAN-based detection scheme, the reconstructor network generates anomalous traffic, and the detector network classifies anomalous traffics and normal traffics in an adversarial manner [22].

In a deep learning detection scheme, architectures that use both a CNN and RNN for IoT network traffic classification are discussed in [23]. The authors show that using various CNN and RNN combinations yield better results of anomaly detection without feature engineering. This research is very similar to my work and supports my claim of high performance.
The use of CNNs for network anomaly detection is explored in [28]. The authors show that deep learning is better than shallow learning due to the non-linearity of network data sets. The research compares simple CNN models to other deep learning structures and shows that other deep learning structures are superior.

In [29] the authors propose a time-series approach using a CNN that regards time-series as a one-dimensional image with the one dimension being temporal. The research is applied to basic streaming data that does not include network data sets. However, the authors show that a CNN architecture can be coupled with a time-series without an RNN.
4.1 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a deep learning algorithm that takes input data, typically an image, and assigns some importance value to parts of the input data to distinguish among different input data. The input data is a two-dimensional grid of values size $J \times J$. For image input data, these values are the pixels in the image. A CNN has multiple layers. Between layers, there is an alternation of convolutions and max pooling, and this process happens to the same initial input data $N$ times with each having a unique filter used during convolutions. Between the initial input data and the first layer is a convolution that uses filters of size $F$, which is less than the dimensions of the input data. These filters create compressed data for the first layer that has a dimension size of $K \times K$, where $K = J - (F - 1)$. Between the first layer and the second layer, max-pooling occurs. Max-pooling down-samples data to provide a better abstraction to help prevent overfitting and reduce the number of learned parameters of the model. Max-pooling can reduce the data by a variable size; however, common max-pooling reduces the data by a factor of two. For example, the second layer would have a dimension size of $L \times L$ where $L = K/2$. This pattern of convolutions followed by max-pooling continues until the original input data is sufficiently reduced. All $N$ resulting compressed data is then fed into a fully connected layer to compare results between the different filtering processes. After the fully connected layer is an output
Figure 1: Diagram of a standard convolutional neural network
layer with abstracted features from the original input data. Figure 1 shows the architecture of a simple CNN.

### 4.2 Residual Neural Network

A Residual Neural Network (ResNet) is a deep learning modular architecture with multiple layers that involves skip connections. An example of the modular architecture can be seen in Figure 2. The skip connections are shortcuts between layers that aren’t usually connected. The reason for these skip connections is to solve the vanishing gradient problem, which is where the updated gradient for a layer in a model can be very small and not cause any change and thus learning in the weights in the layer. The vanishing gradient problem is a well-known issue with deep learning models. The ResNet architecture can be applied to various multi-layer neural networks to help increase depth while mitigating the absence of learning via potential vanishing gradients. The ResNet architecture consists of a modular building block design where for every two layers with weights, there is a skip connection. The skip connection is typically an identity mapping from before the two layers that is added to the output after the two layers. In the example here, we also see the rectified linear unit (ReLU) activation function, which is a commonly used activation function that transforms the weighted input for each layer into the output of that particular input. When looking at the layers more granularly, the ReLU transforms the summed weighted input of a node in a layer to the activation of the node. The ReLU also helps with overcoming the vanishing gradient problem as it is a piecewise linear function that outputs the input if the input is positive and otherwise outputs zero. The ReLU also has other
benefits such as sparse representation within model learning and simple computation due to it really only requiring a max function such as:

```python
def relu(x):
    return max(0, x)
```
Figure 2: Diagram of residual neural network building block
4.3 Recurrent Neural Network

A Recurrent Neural Network (RNN) is a deep learning algorithm that deals with a series type of input instead of the traditional fixed size data input that general neural networks use. The series input for an RNN has no limit on the size of each item in the series; however, the inputs all have to be the same size as the inputs share parameters to gain better insights. Instead of making generalities about a single input data, oftentimes a vector, an RNN looks at how a series of inputs relate to one another. This is represented by a hidden state vector that represents the inputs and outputs among the items in the series. The hidden state deals with both encoding the input data and decoding the output data using previous series item information and the current series item input data. In addition to learning relationships among items in a series, RNNs learn while training, just like other neural networks. RNNs can be made deep with three possible methods [26], including multiple hidden states used per item in the series. RNNs can also be bidirectional where a second hidden state is used that moves in the opposite direction of the initial hidden state. Modern RNNs can use a long short-term memory (LSTM) architecture as well. The LSTM architecture uses cells that are responsible for keeping track of dependencies among portions of the input. These cells are the housing for the hidden state vectors and help with the vanishing gradient problem common with basic deep learning architectures. Diagrams of a normal RNN and bidirectional RNN can be found in Figure 3. In Figure 3, each $h_t$ represents a forward-moving hidden state vector, $g_t$ represents a backward-moving hidden state vector, and $x_t$ and $y_t$ represent the inputs and outputs, respectively.
Figure 3: Diagrams of a basic recurrent neural network and a bidirectional recurrent neural network
4.4 Attention Layer Architecture for RNNs

In a standard RNN algorithm, there exists a problem where longer input sequences make it harder for the hidden state vector to capture and retain earlier inputs and subsequent updates to those inputs. Essentially the more updates made to the hidden state vector, the higher likelihood that information on earlier states is lost. This challenge is solved using an attention architecture proposed by Bahdanau et al in [27]. The attention architecture makes the final output a function of all hidden states. Each decoder output has a distinct vector produced by the attention architecture where the encoder hidden states are represented with distinct weights given to the different encoder hidden states. The output step for an attention layer architecture has a distinct context vector that is a sum product of the attention created weights and hidden states. The attention mechanism is attached to an RNN as an added layer, hence why it is referred to as the attention layer. A couple of drawbacks of using attention are the heavy computations needed for all of the created context vectors and the lack of parallelization since the nature of the hidden state vector requires sequential processing. An example of an attention layer architecture is seen in Figure 4, where each $h_t$ represents a hidden state, $S_t$ represents the summation from all the hidden states, and $x_t$ and $y_t$ represent the inputs and outputs, respectively. Details on the power of attention can be found in [25].
Figure 4: Diagram of an attention layer architecture
4.5 Combining Neural Networks in a Model

Individual neural network architectures can be combined to create more sophisticated models that can give much better insight from input data. For instance, consider analyzing a video for when a live person appears in the video and not just a static image of a person. A combination of a CNN and RNN can be used to determine if a live person appears. The CNN portion could recognize in each frame of the video whether or not the frame contains a person. The RNN portion could then look at the series of frames to determine if the person in each frame is moving or static. Using just the CNN could result in false positives, and using just the RNN could only show that frames were different without identifying a person within each frame. These more complex architectures have component architectures that are modular as neural networks output vectors that can be adjusted to fit input parameters for other neural networks.
CHAPTER V

SUPERB MODEL

5.1 Model Design

The proposed model for SuperB is a deep learning network made of a nine-layer residual neural network (ResNet) with an added fully connected layer that feeds into two bidirectional recurrent neural network (RNN) layers 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 the 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 the model. Figure 5 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 5, normal behavior is labeled as P1 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, 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 $b \times t$ conversations. The conversations are each resized to dimensions of the number of features. For the 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 two layers are the RNN. In the first layer 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 second 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 creates weights on more relevant
input features so that more attention is given to them while extracting patterns from the data. The attention layer ensures that important features are not lost as they can be with a traditional RNN that has complex data as its input. These two 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 5, this classification is split into $P_{\text{Normal}}$ and $Others$, as $P_{\text{Normal}}$ is the selected normal behavior used in the testing.

The number of layers in the SuperB model was established from testing various configurations of layers for the ResNet and RNN. ResNets of size 6, 8, 10, and 12 were used in testing. It was discovered that using 10 layers gave optimal results. Initially, the RNN had only two layers consisting of the bidirectional LSTM cells. The additional attention layer was added to better capture all of the hidden states within the RNN. Thus the final count for layers was set at 16.
Figure 5: Diagram of the SuperB model architecture
The traffic data format is a subset of features found in the conversation view of Wireshark. This can be seen in Table 1.

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.
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 the research to serve as padding to reach a total column count of 98 columns to match parameters in the 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. Using more visited addresses may improve overall results and is discussed in Future Directions. These nominal data vectors comprise the columns within the input data format for the 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 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 1.
Table 2: Identification of collected data

<table>
<thead>
<tr>
<th>Label</th>
<th>Source</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>CICIDS2017</td>
<td>Ubu 16.4 64b</td>
</tr>
<tr>
<td>P2</td>
<td>CICIDS2017</td>
<td>Ubu 14.4 32b</td>
</tr>
<tr>
<td>P3</td>
<td>CICIDS2017</td>
<td>Ubu 16.4 32b</td>
</tr>
<tr>
<td>P4</td>
<td>CICIDS2017</td>
<td>Ubu 14.4 64b</td>
</tr>
<tr>
<td>P5</td>
<td>CICIDS2017</td>
<td>Win 10 Pro 32b</td>
</tr>
<tr>
<td>P6</td>
<td>CICIDS2017</td>
<td>Win 10 64b</td>
</tr>
<tr>
<td>P7</td>
<td>CICIDS2017</td>
<td>Mac OS X</td>
</tr>
<tr>
<td>P8</td>
<td>OfficeB</td>
<td>Win 10 Enterprise 64b</td>
</tr>
<tr>
<td>P9</td>
<td>OfficeB</td>
<td>Win 10 Enterprise 64b</td>
</tr>
<tr>
<td>P10</td>
<td>OfficeB</td>
<td>Win 10 Enterprise 64b</td>
</tr>
<tr>
<td>P11</td>
<td>OfficeB</td>
<td>Win 10 Enterprise 64b</td>
</tr>
<tr>
<td>P12</td>
<td>OfficeA</td>
<td>Win 10 Enterprise 64b</td>
</tr>
<tr>
<td>P13</td>
<td>OfficeA</td>
<td>Win 10 Enterprise 64b</td>
</tr>
<tr>
<td>P14</td>
<td>OfficeA</td>
<td>Win 10 Enterprise 64b</td>
</tr>
<tr>
<td>P15</td>
<td>OfficeA</td>
<td>Win 10 Enterprise 64b</td>
</tr>
<tr>
<td>P16</td>
<td>OfficeA</td>
<td>Win 10 Enterprise 64b</td>
</tr>
</tbody>
</table>

The data used in testing the algorithm is from two sources. The shortened labels for the data can be found in Table 2. 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 [24]. 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 the experiments. All five days of the CICIDS2017 data set are used for the 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, the 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.

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 IIS 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
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 the 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.

5.3 Experiment Design

<table>
<thead>
<tr>
<th>Arrangement</th>
<th>Labeled Normal</th>
<th>Labeled Abnormal</th>
<th>Unlabeled Abnormal</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>OfficeA (P8-P11)</td>
<td>CICIDS2017 (P2, P5, P7)</td>
<td>CICIDS2017 (1 from P3, P4, P6)</td>
</tr>
<tr>
<td>A2</td>
<td>OfficeB (P8-P11)</td>
<td>CICIDS2017 (P1-P3)</td>
<td>CICIDS2017 (1 from P4, P6, P7)</td>
</tr>
<tr>
<td>A3</td>
<td>CICIDS2017 (P1, P5-P7)</td>
<td>OfficeA (P6-P10)</td>
<td>OfficeB (P11)</td>
</tr>
<tr>
<td>A4</td>
<td>OfficeA (P12-P15)</td>
<td>CICIDS2017 (P1-P3)</td>
<td>CICIDS2017 (1 from P4, P6, P7)</td>
</tr>
<tr>
<td>A5</td>
<td>CICIDS2017 (P1, P2, P5, P7)</td>
<td>OfficeA (P12-P14)</td>
<td>OfficeA (1 from P15-P16)</td>
</tr>
<tr>
<td>A6a</td>
<td>OfficeA (P12)</td>
<td>OfficeA (3 from P13-P16)</td>
<td>OfficeA (1 from P13-P16)</td>
</tr>
<tr>
<td>A6b</td>
<td>OfficeA (P13)</td>
<td>OfficeA (3 from P12, P14-P16)</td>
<td>OfficeA (1 from P12, P14-P16)</td>
</tr>
<tr>
<td>A6c</td>
<td>OfficeA (P14)</td>
<td>OfficeA (3 from P12, P13, P15, P16)</td>
<td>OfficeA (1 from P12, P13, P15, P16)</td>
</tr>
<tr>
<td>A6d</td>
<td>OfficeA (P15)</td>
<td>OfficeA (3 from P12-P14, P16)</td>
<td>OfficeA (1 from P12-P14, P16)</td>
</tr>
<tr>
<td>A6e</td>
<td>OfficeA (P16)</td>
<td>OfficeA (3 from P12-P15)</td>
<td>OfficeA (1 from P12-P15)</td>
</tr>
<tr>
<td>A7</td>
<td>OfficeA (P12)</td>
<td>OfficeB (3 from P8-P11) &amp; CICIDS2017 (P1, P5, P7)</td>
<td>OfficeB (1 from P8-P11) &amp; CICIDS2017 (P2) &amp; OfficeA (P15)</td>
</tr>
<tr>
<td>A8</td>
<td>OfficeA (P12)</td>
<td>OfficeB (3 from P8-P11) &amp; CICIDS2017 (P1, P3, P4)</td>
<td>OfficeB (1 from P8-P11) &amp; CICIDS2017 (P2) &amp; OfficeA (P15)</td>
</tr>
</tbody>
</table>

Table 3: Arrangements of data
For the model, I test ten different arrangements using combinations of the CICIDS2017 and the dataset collected from my 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 3.

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 show the strength of the 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 only be used for labeled behavior due to it containing all benign behavior. For arrangement A6, the sixth day was used for testing.
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 data set is required to formally define the number of unique behavior patterns used, which is outside the scope of this paper and addressed in the Future Directions section. Using only CICIDS2017 to test the 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 the model can both distinguish the different users but also can distinguish the different operating systems. The model’s ability to distinguish the different operating systems may be related to the 25 simulated user behaviors found within the seven simulated users I 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, I use a mix of all three data sets. Labeled normal behavior is set to P12 (OfficeA) for both A7 and A8. 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 the random batch selection for each iteration. I train all 24 different models using a time-series $t$ of 5. This time-series amount is enough for distinguishing behavior patterns and small enough for the models to be applicable in real-time analysis of network traffic. I 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.
5.4 Training

Figure 6 shows the general layout of the training for the model. Each step in the training outputs training and testing accuracy. In Figure 6, 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 Pite and the remaining days being training data denoted Pitr 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 also a learning set where one person in the learning set is labeled normal behavior, and the other people in the learning set are labeled 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 $|G_i(S_n \cup S_m)|-1$ as to allow for generalization of abnormal behavior and still have at least one unlabeled test person.

For the experiments, I use one person as labeled normal behavior and three people as labeled abnormal behavior making the size of the learning set four people. The training set I use is dependent on the data set used. Data from the lab consists of eight days of network data per person with seven being used for the training set. Data from the CICIDS2017 data set consists of five days of network data per person; however, the data set behavior is simulated with four being used for the training set. The data sets used in the experiments are discussed in the section on data.
**Result:** Classification Accuracy of Test Data

**Function:**

classify(data1, data2)
{
    data_normal, data_else = process(data1, data2);
    # data_normal_label is [1,0], data_else_label is [0,1]
    return data_normal, data_else;
}

while do_training
    load(training_data, testing_data);
    if results then
        input = results _ training_data;
    else
        input = training_data;
    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);

Figure 6: Algorithm of the deep learning model
### 5.5 Results

<table>
<thead>
<tr>
<th>Test Data</th>
<th>TPR</th>
<th>FPR</th>
<th>Acc</th>
<th>Prec</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1-P8</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A1-P9</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A1-P10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A1-P11</td>
<td>1</td>
<td>0.008</td>
<td>0.996</td>
<td>0.992</td>
<td>0.996</td>
</tr>
<tr>
<td>A2-P8</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A2-P9</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A2-P10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A2-P11</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A3-P1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A3-P5</td>
<td>0.996</td>
<td>0</td>
<td>0.998</td>
<td>1</td>
<td>0.998</td>
</tr>
<tr>
<td>A3-P6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A3-P7</td>
<td>0.99</td>
<td>0.004</td>
<td>0.993</td>
<td>0.996</td>
<td>0.993</td>
</tr>
<tr>
<td>A4-P12</td>
<td>1</td>
<td>0.002</td>
<td>0.999</td>
<td>0.998</td>
<td>0.999</td>
</tr>
<tr>
<td>A4-P13</td>
<td>1</td>
<td>0.002</td>
<td>0.999</td>
<td>0.998</td>
<td>0.999</td>
</tr>
<tr>
<td>A4-P14</td>
<td>1</td>
<td>0.002</td>
<td>0.999</td>
<td>0.998</td>
<td>0.999</td>
</tr>
<tr>
<td>A4-P15</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A5-P1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A5-P2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A5-P5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A5-P7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4: Average performance for A1 - A5 with 10k training steps
<table>
<thead>
<tr>
<th>Test Data</th>
<th>TPR</th>
<th>FPR</th>
<th>Acc</th>
<th>Prec</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A6a</td>
<td>0.988</td>
<td>0.006</td>
<td>0.991</td>
<td>0.996</td>
<td>0.991</td>
</tr>
<tr>
<td>A6b</td>
<td>0.512</td>
<td>0.155</td>
<td>0.678</td>
<td>0.598</td>
<td>0.531</td>
</tr>
<tr>
<td>A6c</td>
<td>0.599</td>
<td>0.65</td>
<td>0.475</td>
<td>0.597</td>
<td>0.476</td>
</tr>
<tr>
<td>A6d</td>
<td>0.97</td>
<td>0.002</td>
<td>0.984</td>
<td>0.999</td>
<td>0.983</td>
</tr>
<tr>
<td>A6e</td>
<td>0.998</td>
<td>0.002</td>
<td>0.994</td>
<td>0.998</td>
<td>0.994</td>
</tr>
</tbody>
</table>

Table 5: Average performance for A6 with 10k training steps

<table>
<thead>
<tr>
<th>Test Data</th>
<th>TPR</th>
<th>FPR</th>
<th>Acc</th>
<th>Prec</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A7-P12-(P8-P11)</td>
<td>0.997</td>
<td>0.002</td>
<td>0.998</td>
<td>0.998</td>
<td>0.997</td>
</tr>
<tr>
<td>A7-P12-P2</td>
<td>1</td>
<td>0.002</td>
<td>0.999</td>
<td>0.998</td>
<td>0.999</td>
</tr>
<tr>
<td>A7-P12-P15</td>
<td>0.93</td>
<td>0.002</td>
<td>0.964</td>
<td>0.998</td>
<td>0.963</td>
</tr>
<tr>
<td>A8-P12-(P8-P11)</td>
<td>0.9935</td>
<td>0.011</td>
<td>0.991</td>
<td>0.989</td>
<td>0.991</td>
</tr>
<tr>
<td>A8-P12-P2</td>
<td>1</td>
<td>0.011</td>
<td>0.995</td>
<td>0.989</td>
<td>0.995</td>
</tr>
<tr>
<td>A8-P12-P15</td>
<td>0.922</td>
<td>0.011</td>
<td>0.956</td>
<td>0.988</td>
<td>0.954</td>
</tr>
</tbody>
</table>

Table 6: Average performance for A7 & A8 with 10k training steps

Figures 7 through 12 represent the experimental results based on two datasets: CICIDS2017 and IIS. Figure 10, Figure 11, and Figure 12 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 7, Figure 8, and Figure 9. 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 4, Table 5, and Table 6. Table 4 has arrangements A1 to A5, Table 5 has the sub-arrangements of A6, and Table 6 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 6, I did a combination of data different from all other arrangements. This combination is done to further show the strength of the model and to remove bias from the testing. In the results found in Table 6, the average of four simulations per sub-arrangement is used. In the selection of both labeled and unlabeled behavior, I picked an arbitrary grouping from the CICIDS2017 and IIS data sets. The column Test Data represents a combination of arrangement-labeled normal-unlabeled abnormal. I 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 the IIS 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, the model is very effective in using live network data.
These results suggest the 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 the model is resistant to the base rate fallacy problem. The 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.

During the experiments, I found some cases that the algorithm does not work well. Figure 8 and Figure 1, 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 the 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 the results below, I ignore A6-P14 and A6-P15.

For the efficiency matter, I 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 seconds for the 10,000 training steps. The experiment showed that SuperB 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.
Figure 7: False positive ratios for A1 - A5
Figure 8: False positive ratios for A6
Figure 9: False positive ratios for A7 & A8
Figure 10: True positive ratios for A1 - A5
Figure 11: True positive ratios for A6
Figure 12: True positive ratios for A7 & A8
5.6 Future Directions

There is further work and experimentation to be done regarding this research. A majority of the behavior data comes from visited web addresses. More experiments are needed where more visited web addresses are used. This is predicted to improve results. To achieve zero false positives, and in turn a perfect accuracy, more data should be used to ensure that any patterns of behavior by users are adequately captured. Some behaviors can span over certain times of the year or month, such as when certain sports are being played, and users have a live stream of the games being played. Other behaviors can be periodic with weeks in between such as when employees interact with their bank on paydays for a company. However, to the best of my knowledge, there are no public network data sets fitting these criteria. More data arrangements can be used in experiments to further test the model as the different arrangements would test the robustness of the model. The CICIDS2017 data set needs a further analysis of the individual behavior patterns among the different machines used in the simulated network data. It has been verified through testing that each operating system has a unique pattern of behavior that can be used in the model. All of these improvements are left open for further research.
CHAPTER VI

CONCLUSION

In this thesis, I 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. This solution is a semi-supervised deep learning approach that showed a high accuracy rate of detecting unlabeled abnormal behavior. To the best of my 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 the experiment showed that SuperB 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 this algorithm.
REFERENCES


