System Architecture Design

For an Intrusion Detection System using a Neural Network

Version 2.0

Submitted in partial fulfillment of the requirements of the degree of MSE

Blake Knedler

CIS 895 – MSE Project

Kansas State University

Table of Contents

1 Introduction

This document will provide the system design information for the PyIDS – a python interpretation of an intrusion detection system. The intrusion detection system is a single component itself but consists of several pieces that work together to perform the required functionality. This document will detail the detailed design of the system using component and interface specifications in the form of the standard UML design language.

2 Architecture

The Intrusion Detection System architecture is a very simple design. The architecture is a layered approach that is event driven. There are three main layers contained within the IDS. The three layers are the Network Traffic Reader, Neural Network, and Recorder. The Network Traffic Reader is a data reading layer of the system. It will take the data from the network card and package it in a way that is useful to the rest of the system. The Neural Network layer of the system is the brains of the system. It will take the data that is read in the Network Traffic Reader layer and make a decision based on backpropogation training or loaded synapse weights. The final layer is the Recorder. The Neural Network layer will communicate to this layer indicating any malicious packets it has received. It is the Recorder layer's responsibility to log that information and notify the user. Since the system architecture is simple in nature, the Recorder also acts as the user interface. It will respond to the user when a start or train sequence is requested and notify the other layers of this information.

3 Component Design

In this section, we will look at the different components of the system and focus on how they interact with each other through interfaces. We will also look at what interfaces the overall system has with external devices and users.

3.1 Component Diagram

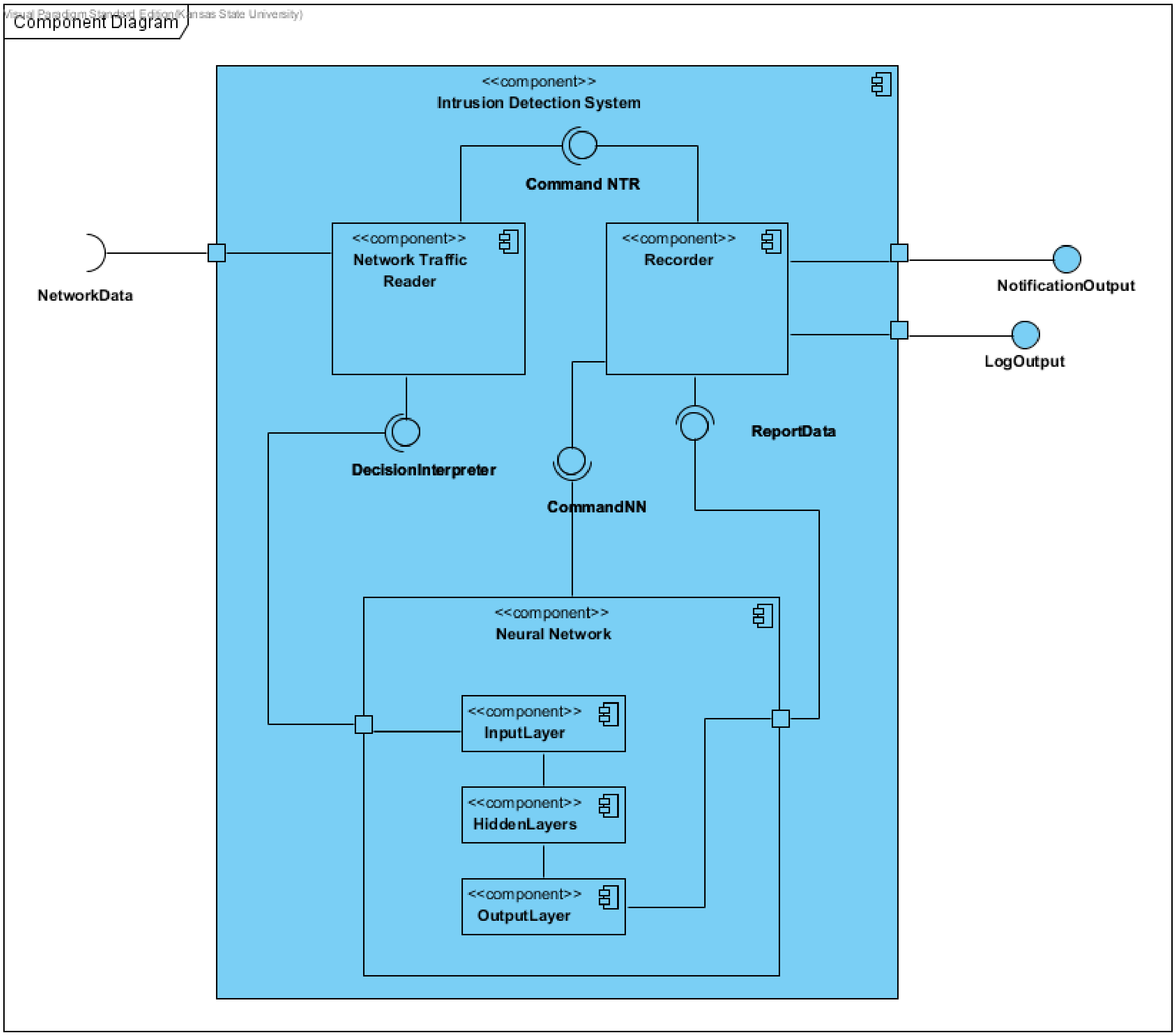

Figure 1. IDS System Component Design

3.2 Component Interface Specification

Figure 1 shows the component diagram of the Intrusion Detection System. There are three main components contained in the overall component of the Intrusion Detection System. There are also three external interfaces to this system which are the NetworkData, which is the data traffic, the NotificationOutput and

LogOutput which are notifications to the user about the data the system read. The Network Traffic Reader component is the component of the Intrusion Detection System that will read the network traffic. It will then pass this data via the DecisionInterpreter connection to the Neural Network component. This Neural Network component consists of three sub-components. These components are each of the layers of the Neural Network component. The InputLayer component will receive the data for the Neural Network component and pass the data on to the HiddenLayer and then to the OutputLayer. The OutputLayer then provides the connection of ReportData to the Recorder component. This connection will be how the Neural Network component passes any decisions of malicious data traffic to the recorder. Any non-malicious traffic can also be passed via this interface as well. The final component of this Intrusion Detection System is the Recorder component. This component is responsible for both logging and notifying the user of the data that it received.

4 Package Design

In this section, we will go into more detail about each of the packages of the system breaking them down by the different pieces and describing what they are doing for the system and how they are playing a role in the system.

4.1 System Context Package Diagram

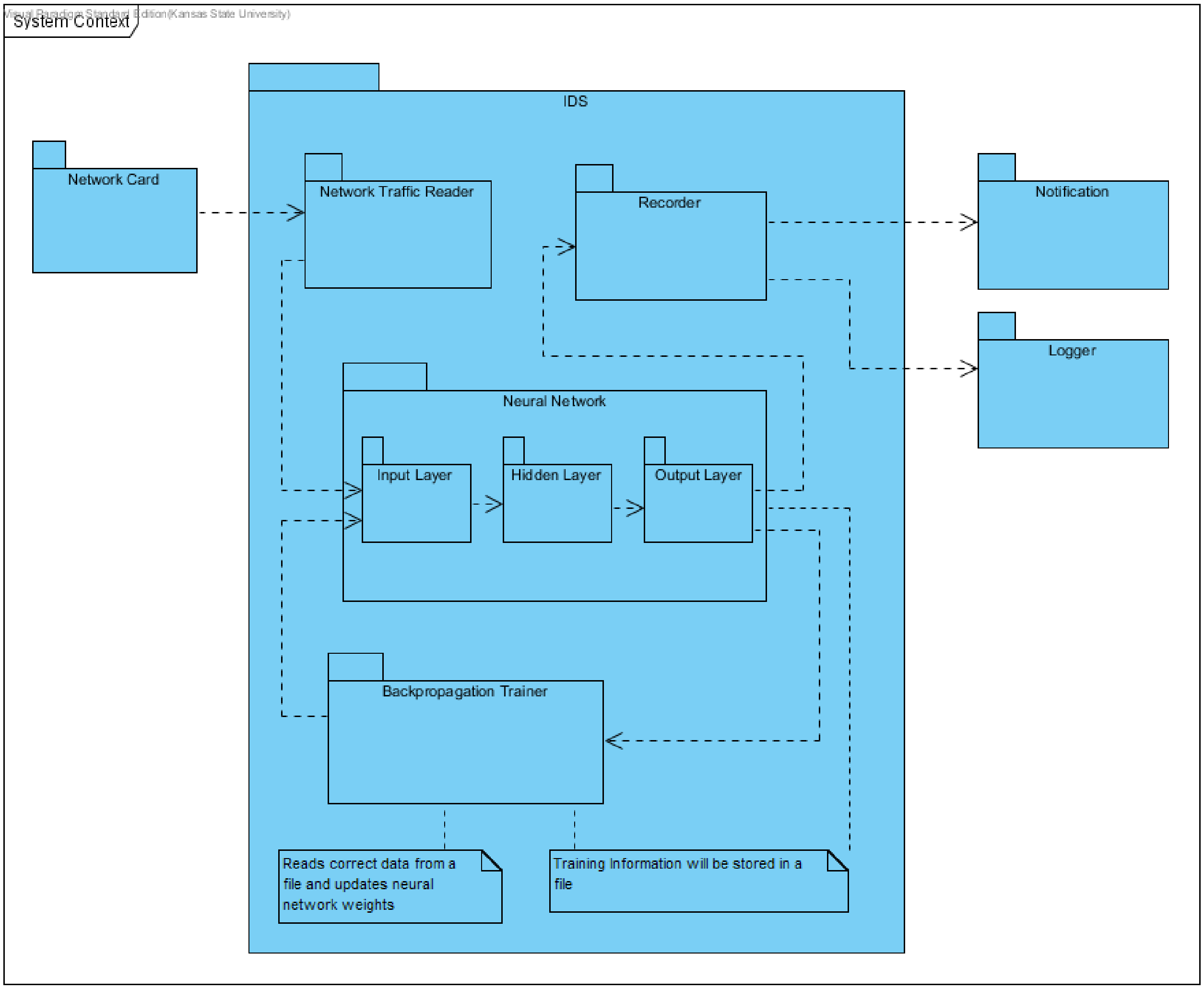
System Context
Visual Paradigm Standard Edition(Kansas State University)
IDS
Network Card
Network Traffic Reader
Recorder
Notification
Logger
Neural Network
Input Layer
Hidden Layer
Output Layer
Backpropagation Trainer
Reads correct data from a file and updates neural network weights
Training Information will be stored in a file

Figure 2. IDS System Package Diagram

4.1.1 Network Traffic Reader

The Network Traffic Reader package is the data reader for the system. The main goal of this package is to read the network traffic off of the network card as quickly as possible. There are no strict time limits set for this since the configuration of the host system can vary. The Network Traffic Reader package will also be responsible for stripping the network packages as they read to obtain the important information useful to the system. This data will be modeled after the training data to allow for the system to make correct decisions on the packets. This package is very simple in nature and really on performs two real functions. To grab the data and get it into a useable format.

4.1.2 Neural Network

The Neural Network package is the brains of the system. The Neural Network package reads the data it receives from the Network Traffic Reader package and makes a decision based on its training data to determine if the data packet is malicious or not. The Neural Network consists of three smaller packages. These packages are the individual layers of the Neural Network that work together to appropriately manipulate the data to come to a conclusion for an answer. The Input Layer package will take the data in and store it in a way that is useful to the Neural Network. The Hidden Layer will receive the data from the Input Layer and pass it to the Output Layer for the decision. In the passing of data from one layer to the next a set of weights are applied and functions are used to manipulate the data to find patterns associated with the trained data. This helps determine what the outcome of the data should be. The Output Layer will receive a single answer about each packet. This answer will be if the data packet is malicious or not. The Output Layer is also in charge of letting the Recorder package know if a malicious packet has been detected. The overall goal of the Neural Network is to make a decision on the data packet coming in and letting the Recorder package know if it has found a malicious packet.

4.1.3 Backpropogation Trainer

The Backpropogation Trainer package is in charge of training the Neural Network package. This is an extremely crucial part of the system that is enforced prior to running the system to capture live data. The system will have a file containing the training data which will be used by this package. The package will feed the Neural Network package the data and assess the correctness of the Neural Network. The package will continue to train the Neural Network package until it has reach a certain level of validity with the answers. The Backpropogation Trainer will also allow the Neural Network to store a file of previously trained weights. This allows the system to only need to be trained once and to reuse that training information for future sessions.

4.1.4 Recorder

The Recorder package is the user interface portion of the system. This package is in charge of reporting any malicious packets found by the system and logging that information. The Recorder will also handle the other small user interface connections that are needed to start, stop, and train the system via a user interface. The main task of the Recorder package is to take the data from the Neural Network and inform the user of the malicious data that was found. This data will also be stored in a log file for future reference to the user. Since there is not a user interface packet due to the small amount of user interface needs, the Recorder will act as the user interface. This package will be in charge of taking the user input to start the system, stop the system, train, etc. It will then relay that information to the rest of the system to begin the task that was requested.

5 Class Design

In this section, we will look briefly at a class design diagram of the three components of the system. Since this is a high level architecture document, this section will not completely detail each of these classes.

5.1 Class Design Diagram

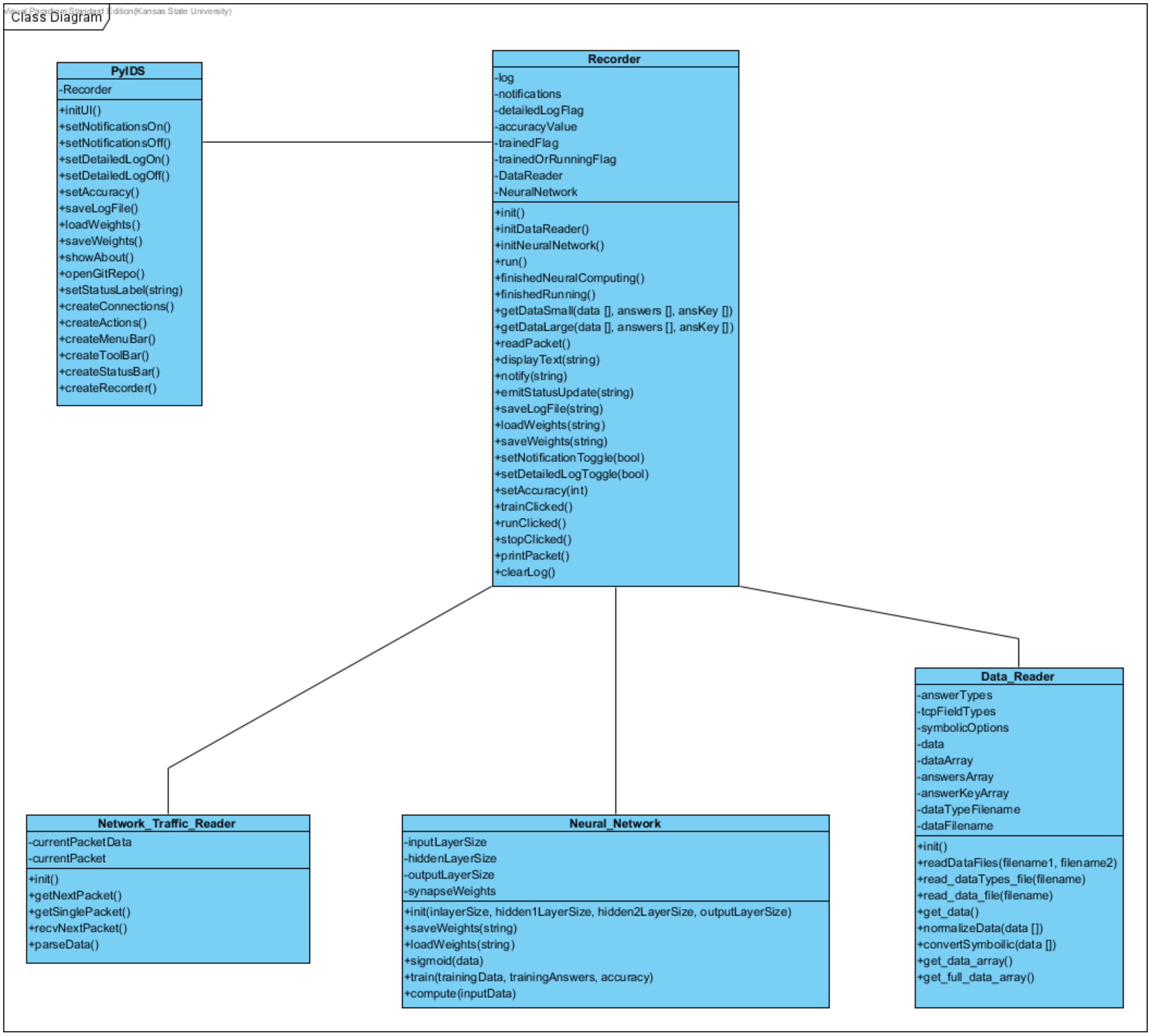

Figure 3. IDS System Class Diagrams

Figure 3 shows a class diagram of the three main components of the system. As mentioned previously, these three components are the Recorder, the Neural Network, and the Network Traffic Reader.

5.1.1 Network Traffic Reader

Name:	init
Purpose	This function will initialized the Network Traffic Reader
Inputs:	None
Outputs:	et
Pre-Conditions:	None
Post-Conditions:	The Network Traffic Reader will be initialized.

Name:	getNextPacket
Purpose	This function will get the next packet on the list.
Inputs:	None
Outputs:	currentPacketData | string of data
Pre-Conditions:	The current packet data must be set.
Post-Conditions:	None

Name:	getSinglePacket
Purpose	This function will get a single new packet.
Inputs:	None
Outputs:	currentPacketData | string of data
Pre-Conditions:	None
Post-Conditions:	The current packet data will be set and returned.

Name:	recvNextPacket
Purpose	This function will grab the newest packet off of the network card.
Inputs:	None
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The current packet will be set.

Name:	parseData
Purpose	This function will parse the data out of the current packet.
Inputs:	currentacket | string of data
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The current packet data will be set.

5.1.2 Recorder

Name:	init
Purpose	This function will initializes the Recorder.
Inputs:	None
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	Recorder is initialized.

Name:	initDataReader
Purpose	This function will initialize the data reader part of the system.
Inputs:	None
Outputs:	None
Pre-Conditions:	The Recorder is initialized.
Post-Conditions:	The system will have the data and be considered initialized.

Name:	initNeuralNetwork
Purpose	This function will initialize the neural network part of the system.
Inputs:	None
Outputs:	None
Pre-Conditions:	The Recorder is initialized.
Post-Conditions:	The system will have initilized the neural network and be considered trained.

Name:	run
Purpose	This function will tell the system to begin running.
Inputs:	None
Outputs:	None
Pre-Conditions:	The Neural Network must be trained.
Post-Conditions:	The system will be operating.

Name:	finishedNeuralComputing
Purpose	This function will determine if the system should begin running or not after training.
Inputs:	None
Outputs:	None
Pre-Conditions:	The Neural Network must be trained.
Post-Conditions:	The system will be considered trained.

Name:	finishedRunning
Purpose	This function will perofrm cleanup on the completion of a run.
Inputs:	None
Outputs:	None
Pre-Conditions:	The system is running.
Post-Conditions:	The system is stopped.

Name:	getDataSmall
Purpose	This function stores the small data set
Inputs:	data | list of values; answers | list of values; ansKey | list of values
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The data will be stored in the Recorder.

Name:	getDataLarge
Purpose	This function stores the large data set
Inputs:	data | list of values; answers | list of values; ansKey | list of values
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The data will be stored in the Recorder.

Name:	readPacket
Purpose	This function reads a single new packet.
Inputs:	None
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The single packet will be sent to the log and notifications.

Name:	displayText
Purpose	This function will print the text to the log and store it to be printed later.
Inputs:	info | string
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The data will be stored and printed to the log.

Name:	notify
Purpose	This function will print the notification string.
Inputs:	info | string
Outputs:	Notification
Pre-Conditions:	None
Post-Conditions:	The info will be displayed in a notification.

Name:	emitStatusUpdate
Purpose	This function will tell the status bar to update the status.
Inputs:	status | string
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The status bar will be updated with new status.

Name:	saveLogFile
Purpose	This function will save the log data to a file.
Inputs:	filename | string
Outputs:	Log File
Pre-Conditions:	None
Post-Conditions:	The log file will be written out.

Name:	loadWeights
Purpose	The function will load the currently saved synapse weights into the Neural Network.
Inputs:	filename | string
Outputs:	None
Pre-Conditions:	The UI is initialized and saved weights must exist.
Post-Conditions:	The Neural Network will be trained.

Name:	saveWeights
Purpose	The function will save the current weights to a file.
Inputs:	filename | string
Outputs:	None
Pre-Conditions:	The UI is initialized.
Post-Conditions:	There will be a save weights file.

Name:	setNotificationToggle
Purpose	This function will set the flag to show or not show notifications.
Inputs:	flag | bool
Outputs:	None
Pre-Conditions:	The UI is initialized.
Post-Conditions:	The notifications flag will be set.

Name:	setDetailedLogToggle
Purpose	This function will set the flag to show or not show the detailed log.
Inputs:	flag | bool
Outputs:	None
Pre-Conditions:	The UI is initialized.
Post-Conditions:	The detailed log flag will be set.

Name:	setAccuracy
Purpose	This function will set the current minimum accuracy requirement.
Inputs:	value | integer
Outputs:	None
Pre-Conditions:	The UI is initialized.
Post-Conditions:	The minimum accuracy value will be set.

Name:	trainClicked
Purpose	This function will start training the system
Inputs:	None
Outputs:	None
Pre-Conditions:	The UI is initialized.
Post-Conditions:	The system will be trained.

Name:	runClicked
Purpose	This function will start running the system
Inputs:	None
Outputs:	None
Pre-Conditions:	The system is trained.
Post-Conditions:	The system will be running.

Name:	stopClicked
Purpose	This function will tell the system to stop running.
Inputs:	None
Outputs:	None
Pre-Conditions:	The system must be running.
Post-Conditions:	The system must not be running.

Name:	printPacket
Purpose	This function will get a new packet and print it.
Inputs:	None
Outputs:	None
Pre-Conditions:	The UI is initialized.
Post-Conditions:	A new packet will be passed to the readPacket function.

Name:	clearLog
Purpose	This function will clear the displayed log.
Inputs:	None
Outputs:	None
Pre-Conditions:	The UI is initialized.
Post-Conditions:	The log will be cleared.

5.1.3 Neural Network

Name:	init
Purpose	This function will set the size for the number of the layer nodes.
Inputs:	inputLayerSize | integer; hiddenLayer1Size | integer; hiddenLayer2Size | integer; outputLayerSize | integer
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The input layer size will exist.

Name:	saveWeights
Purpose	This function will save the weights.
Inputs:	filename | string
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The weights of the system will be saved.

Name:	loadWeights
Purpose	This function will load the weights.
Inputs:	filename | string
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The weights will be loaded.

Name:	sigmoid
Purpose	This function will perform a sigmoid function on the data it is given.
Inputs:	data | matrix of integers
Outputs:	data | matrix of integers
Pre-Conditions:	None
Post-Conditions:	None

Name:	train
Purpose	This function will train the Neural Network.
Inputs:	trainingData | matrix of integers trainingAnswers | matrix of integers accuracy | integer
Outputs:	data | matrix of integers
Pre-Conditions:	Training data must have been read into the Neural Network.
Post-Conditions:	The Neural Network will be considered trained.

Name:	compute
Purpose	This function will compute the decision for a signle set of data.
Inputs:	data | matrix of integers
Outputs:	answer | integer
Pre-Conditions:	The Neural Network must be trained.
Post-Conditions:	None

5.1.4 Data Reader

Name:	init
Purpose	This function will initialized the Data Reader.
Inputs:	None
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The Data Reader will be initialized.

Name:	readDataFiles
Purpose	This function will call the methods to read the data files.
Inputs:	dataTypeFile | string; dataFile | string
Outputs:	None
Pre-Conditions:	The Data Reader is initialized.
Post-Conditions:	The data files have been read into memory.

Name:	read_dataTypes_file
Purpose	This function will read the dataTypes file and store the values needed into data structures.
Inputs:	filename | string
Outputs:	None
Pre-Conditions:	The Data Reader is initialized.
Post-Conditions:	The data file has been read into memory.

Name:	read_data_file
Purpose	This function will read the data file and store the values needed into data structures.
Inputs:	filename | string
Outputs:	None
Pre-Conditions:	The Data Reader is initialized.
Post-Conditions:	The data file has been read into memory.

Name:	get_data
Purpose	This function returns the current data.
Inputs:	None
Outputs:	data | list of strings
Pre-Conditions:	The data has been read into memeory.
Post-Conditions:	None

Name:	normalizeData
Purpose	This function returns the current data normalized.
Inputs:	data | list of strings
Outputs:	data | list of strings
Pre-Conditions:	The data has been read into memeory.
Post-Conditions:	None

Name:	convertSymbolic
Purpose	This function returns symbolic values as the actual numerical value.
Inputs:	data | list of strings
Outputs:	data | list of strings
Pre-Conditions:	The data has been read into memeory.
Post-Conditions:	None

Name:	get_data_array
Purpose	This function parses out the useful data from the data in memory.
Inputs:	None
Outputs:	data | list of strings
Pre-Conditions:	None
Post-Conditions:	The data has been read into memeory.

Name:	get_full_data_array
Purpose	This function parses out the useful data from the data in memory.
Inputs:	None
Outputs:	data | list of strings
Pre-Conditions:	None
Post-Conditions:	The data has been read into memeory.

5.1.5 PyIDS (UI)

Name:	initUI
Purpose	This function will initialize the User Interface (UI) portion of the system.
Inputs:	None
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The UI will be initialized and can be displayed.

Name:	setNotificationsOn
Purpose	This function will tell the Recorder to turn on the notifications setting.
Inputs:	None
Outputs:	None
Pre-Conditions:	The UI is initialized.
Post-Conditions:	Notifications can now be displayed.

Name:	setNotificationsOff
Purpose	This function will tell the Recorder to turn off the notifications setting.
Inputs:	None
Outputs:	None
Pre-Conditions:	The UI is initialized.
Post-Conditions:	Notifications can now not be displayed.

Name:	setDetailedLogOn
Purpose	This function will tell the Recorder to turn on the detailed log setting.
Inputs:	None
Outputs:	None
Pre-Conditions:	The UI is initialized.
Post-Conditions:	The detailed log setting will be on.

Name:	setDetailedLogOff
Purpose	This function will tell the Recorder to turn off the detailed log setting.
Inputs:	None
Outputs:	None
Pre-Conditions:	The UI is initialized.
Post-Conditions:	The detailed log setting will be off.

Name:	setAccuracy
Purpose	This function will tell the Recorder the minimum accuracy requirement for the Neural Network.
Inputs:	None
Outputs:	None
Pre-Conditions:	The UI is initialized.
Post-Conditions:	The Neural Network accuracy setting will be changed

Name:	saveLogFile
Purpose	This function will save the log to a file.
Inputs:	None
Outputs:	Log File
Pre-Conditions:	The UI is initialized.
Post-Conditions:	The log will be saved to a file.

Name:	loadWeights
Purpose	This funtion will load a weights file.
Inputs:	None
Outputs:	None
Pre-Conditions:	The UI is initialized.
Post-Conditions:	The weights file will be loaded.

Name:	saveWeights
Purpose	This funtion will save the weights to a file.
Inputs:	None
Outputs:	Weights File
Pre-Conditions:	The UI is initialized.
Post-Conditions:	The weights will be saved to a file.

Name:	showAbout
Purpose	This function will show the about window.
Inputs:	None
Outputs:	The about window will be up.
Pre-Conditions:	The UI is initialized.
Post-Conditions:	The about window will be shown.

Name:	openGitRepo
Purpose	This function will take the user to the GitHub page.
Inputs:	None
Outputs:	The GiyHub page will be opened.
Pre-Conditions:	The UI is initialized.
Post-Conditions:	The GitHub page will be shown.

Name:	setStatusLabel
Purpose	This function will change the status on the status bar.
Inputs:	status | string
Outputs:	None
Pre-Conditions:	The UI is initialized.
Post-Conditions:	The status label on the status bar is updated.

Name:	createConnections
Purpose	This function will setup connections for when actions are triggered.
Inputs:	None
Outputs:	None
Pre-Conditions:	The actions have been created.
Post-Conditions:	The connections have been created.

Name:	createActions
Purpose	This function will create actions that the user can perform.
Inputs:	None
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The actions have been created.

Name:	createMenuBar
Purpose	This function will create the menu bar.
Inputs:	None
Outputs:	None
Pre-Conditions:	The actions have been created.
Post-Conditions:	The menu bar has been created.

Name:	createToolBar
Purpose	This function will create the tool bar.
Inputs:	None
Outputs:	None
Pre-Conditions:	The actions have been created.
Post-Conditions:	The tool bar has been created.

Name:	createStatusBar
Purpose	This function will create the status bar.
Inputs:	None
Outputs:	None
Pre-Conditions:	The actions have been created.
Post-Conditions:	The status bar has been created.

Name:	createRecorder
Purpose	This function will create the Recorder.
Inputs:	None
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The Recorder has been created.

6 Sequence Design

In this section, we will look at a couple of the main operating sequences and how system communicates between the different internal components and also to any external user or device.

6.1 Operating Sequence Diagram

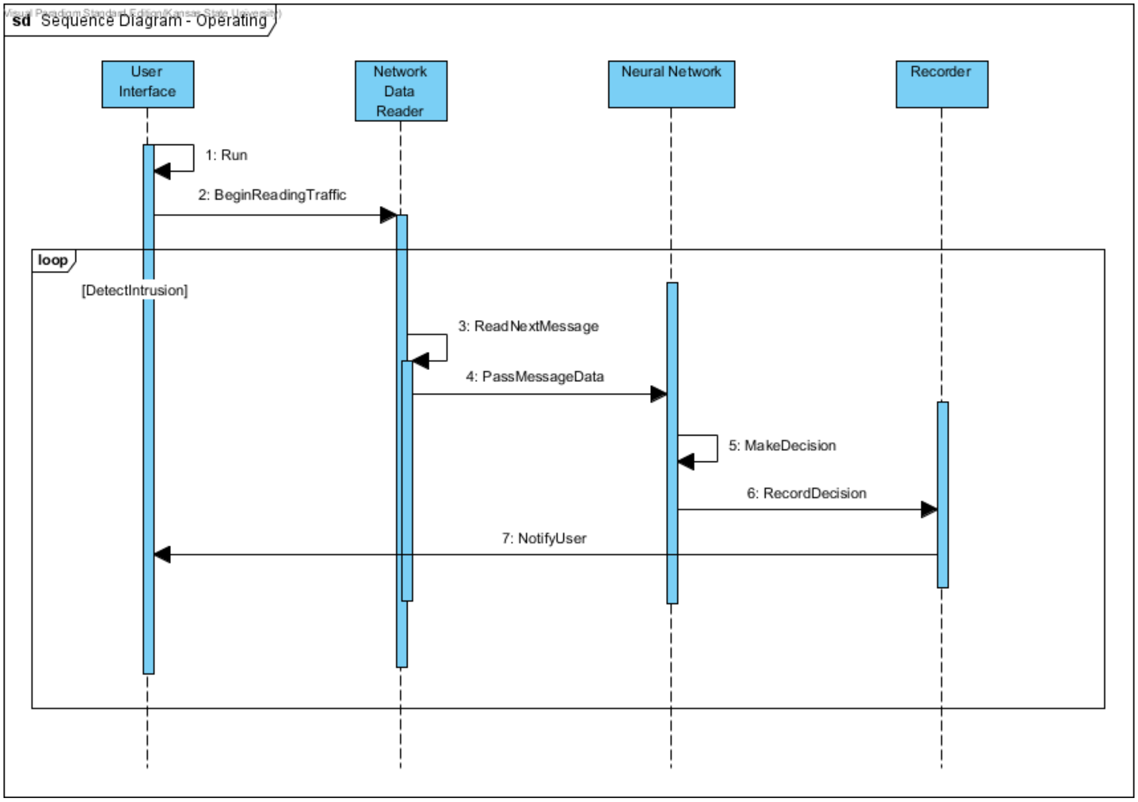

Figure 4. IDS System Operating Sequence Diagram

Figure 4 shows a sequence diagram of the main operating sequence for the Intrusion Detection System. The sequence begins with the user selecting to begin running the system. The user interface is separated in the diagram to help distinguish between it and the Recorder functionality. As mentioned previously, these two aspects will be handled by the Recorder. The User Interface notifies the Network Data Reader to begin reading network traffic. As each message is read, the Network Data Reader will pass the message data to the Neural Network. The Neural Network will then make a decision about the data packet and notify the Recorder of malicious packets. The Recorder will then notify the user by a notification system and a logging system. This process is an iterative process for each message read by the Network Data Reader.

6.2 Train Sequence Diagram

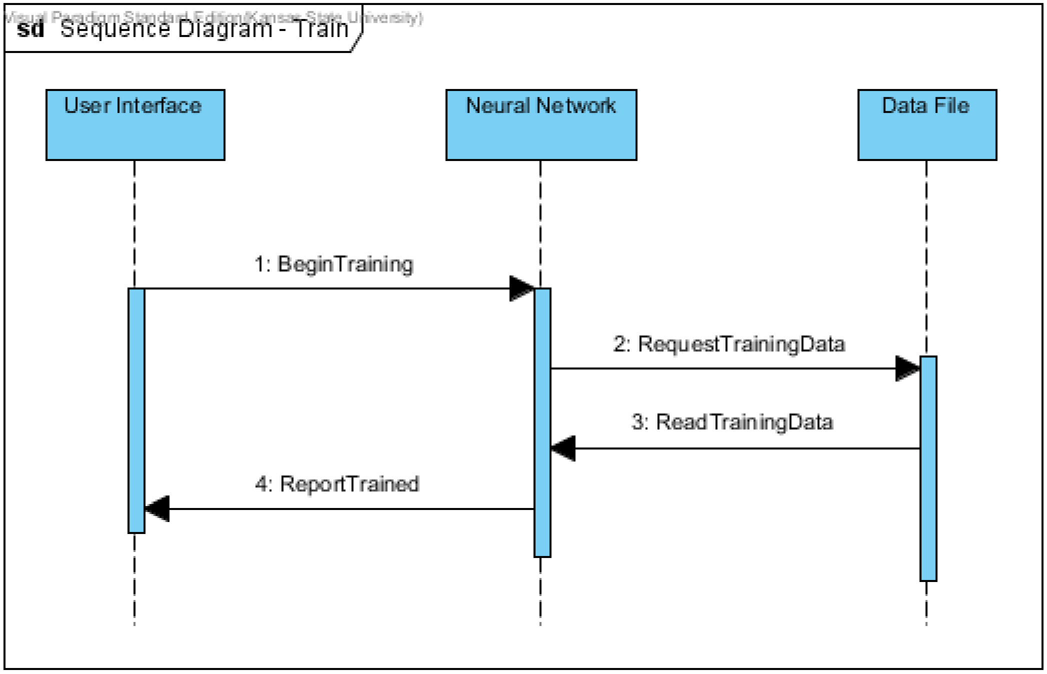

Figure 5. IDS System Train Sequence Diagram

Figure 5 shows a sequence diagram of the training scenario. This sequence begins with the user selecting to begin training. The User Interface notifies the Neural Network to begin training. The Neural Network will read the Data File containing the training data. It will then evaluate its performance of the training information and notify the user that it has now been trained and is ready to start.

7 Formal Requirements Specification Design

```
--
-- PyIDS Formal Specification
--
-- pyIDS_fs.use
--
-- This is a formal specification for the
-- Python Intrusion Detection System (PyIDS).
--
--
-- Author: Blake Knedler
-- Date: September 11, 2016
--

model PyIDS

-- ===========================================================
-- E N U M E R A T I O N S
-- ===========================================================

-- Enumeration list of packet types
enum PacketType {Valid, Malicious}



--
-- C L A S S E S
--

-- Packet class
-- - basic definition of what a packet will be
class Packet
attributes
     data : String
     type : PacketType
end

-- PacketReader class
-- - will read the packets as they are received
class PacketReader
attributes
     currentPacket : Packet
operations
     readPacket(p : Packet) : Boolean =
          self.currentPacket = p
end

-- NeuralNetwork class
-- - will make decisions based on the validity of the packet
class NeuralNetwork
attributes
     decision : Boolean
operations
     -- Decision will be true if the packet is malicious
     makeDecision(p : Packet) : Boolean =
          if (p.type = #Malicious) then
               true
          else
               false
          endif
end

-- Notifier class
-- - will notify user of the malicious packets
class Notifier
attributes
     notification : String
     numberOfMaliciousPackets : Integer
end


--
-- A S S O C I A T I O N S
--

-- Association relating the reader (PacketReader)
--   to reading Packets
association ReadPackets between
     PacketReader [1] role reader
     Packet [*] role packet
end

-- Association relating the reader (PacketReader)
--   and decision maker (NeuralNetwork)
association DetermineDecision between
     PacketReader [1] role reader
     NeuralNetwork [1] role decider
end
```

```
-- Association relating the decision maker (NeuralNetwork)
--   to the notificaiton system (Notifier)
association Notifications between
     NeuralNetwork [1] role decider
     Notifier [1] role notifier
end



--
-- C O N S T R A I N T S
--

constraints

-- There is only one Packet Reader
context PacketReader
     inv Only_One_PacketReader:
          PacketReader.allInstances->size = 1

-- There is only one Neural Network
context NeuralNetwork
     inv Only_One_NeuralNetwork:
          NeuralNetwork.allInstances->size = 1

-- There is only one Notification system
context Notifier
     inv Only_One_Notifier:
          Notifier.allInstances->size = 1

-- Invariant to report a malicious packet
context p:Packet
     inv ReportMalicousPacket:
          (NeuralNetwork.allInstances)->forAll(nn |
          ( if nn.decision = true then
               nn.notifier.notification = p.data and
               nn.notifier.numberOfMaliciousPackets =
nn.notifier.numberOfMaliciousPackets + 1
          else
               nn.notifier.numberOfMaliciousPackets =
nn.notifier.numberOfMaliciousPackets
          endif ))
```