INTRUSION DETECTION SYSTEM USING A NEURAL NETWORK

PHASE TWO PRESENTATION

BLAKE KNEDLER

AGENDA

- Action Items
- Project Vision and Plan Updates
- Formal Requirements Specification
- Architecture Design
- Test Plan
- Formal Technical Inspection Checklist
- Risks and Concerns
- Phase Three Plan
- Architecture Prototype Demonstration
- Questions and Comments

ACTION ITEMS

- Updated Vision Document for accuracy documentation
- Updated Project Plan for correct COCOMO chart
- Added risks for KDD99 Dataset

PROJECT VISION AND PLAN UPDATES

- Vision Document Updates
  - Updated to mention that I plan to achieve 85% accuracy.
  - Noted that all false negatives and false positives hurt that accuracy
- Project Plan Updates
  - Updated COCOMO chart show on the next slide

PROJECT VISION AND PLAN UPDATES

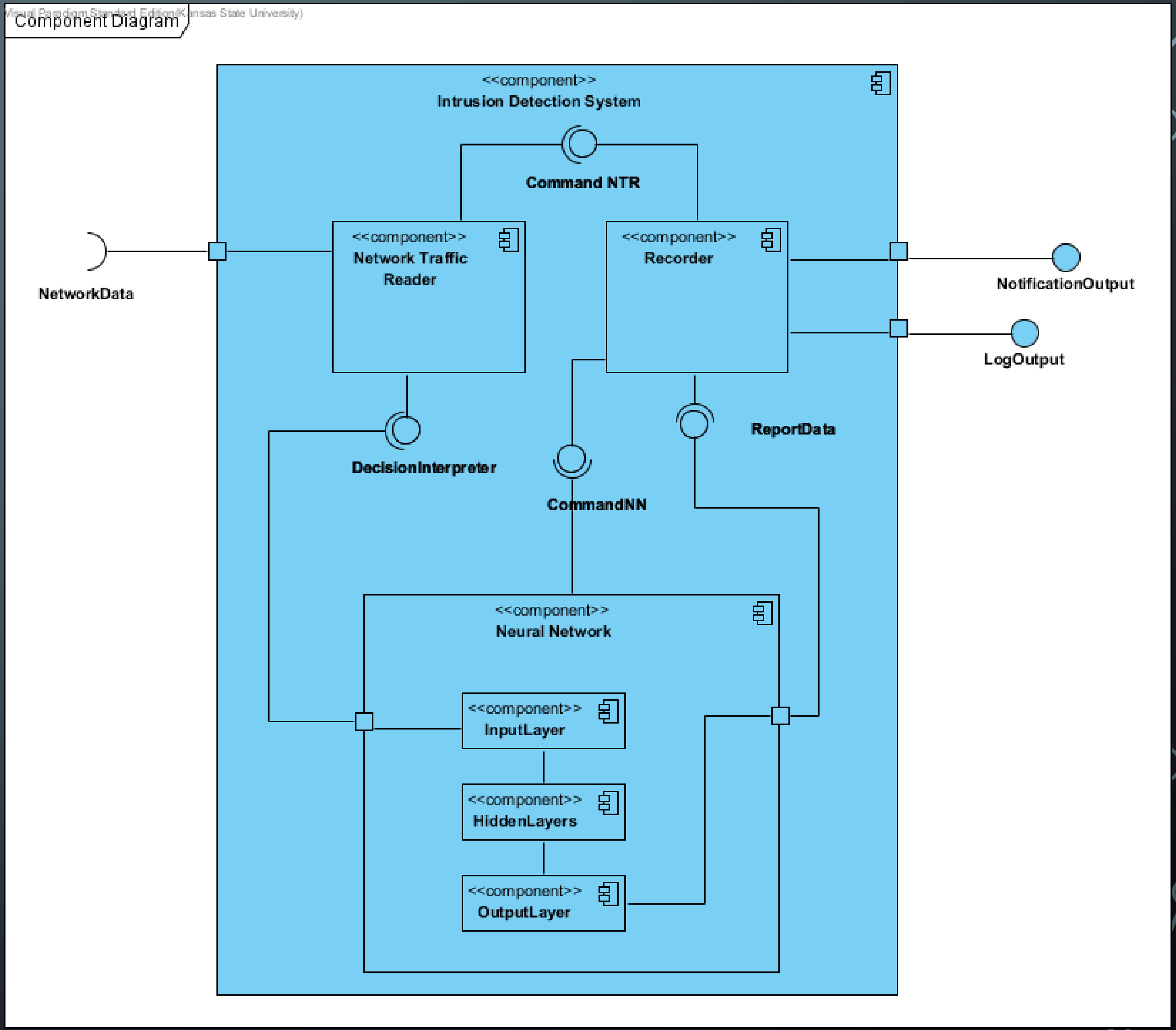
Cost Drivers	Ratings						Chosen Result
	Very Low	Low	Nominal	High	Very High	Extra High	
Product attributes							
Required software reliability	0.75	0.88	1	1.15	1.4		1.15
Size of application database		0.94	1	1.08	1.16		1
Complexity of the product	0.7	0.85	1	1.15	1.3	1.65	1.3
Hardware attributes							
Run-time performance constraints			1	1.11	1.3	1.66	1.3
Memory constraints			1	1.06	1.21	1.56	1
Volatility of the virtual machine environment		0.87	1	1.15	1.3		1
Required turnabout time		0.87	1	1.07	1.15		1.07
Personnel attributes							
Analyst capability	1.46	1.19	1	0.86	0.71		0.86
Applications experience	1.29	1.13	1	0.91	0.82		0.91
Software engineer capability	1.42	1.17	1	0.86	0.7		0.7
Virtual machine experience	1.21	1.1	1	0.9			1
Programming language experience	1.14	1.07	1	0.95			0.95
Project attributes							
Application of software engineering methods	1.24	1.1	1	0.91	0.82		1
Use of software tools	1.24	1.1	1	0.91	0.83		0.83
Required development schedule	0.82	0.96	1	1.04	1.1		1.04
TOTAL EAF	**0.9342**						

FORMAL REQUIREMENTS SPECIFICATION

- Developed using USE
  - Four main classes:
    - Packet
    - PacketReader (Network Traffic Reader)
    - NeuralNetwork
    - Notifier (Recorder)
  - Operations for the PacketReader and NeuralNetwork include
    - readPacket
    - makeDecision

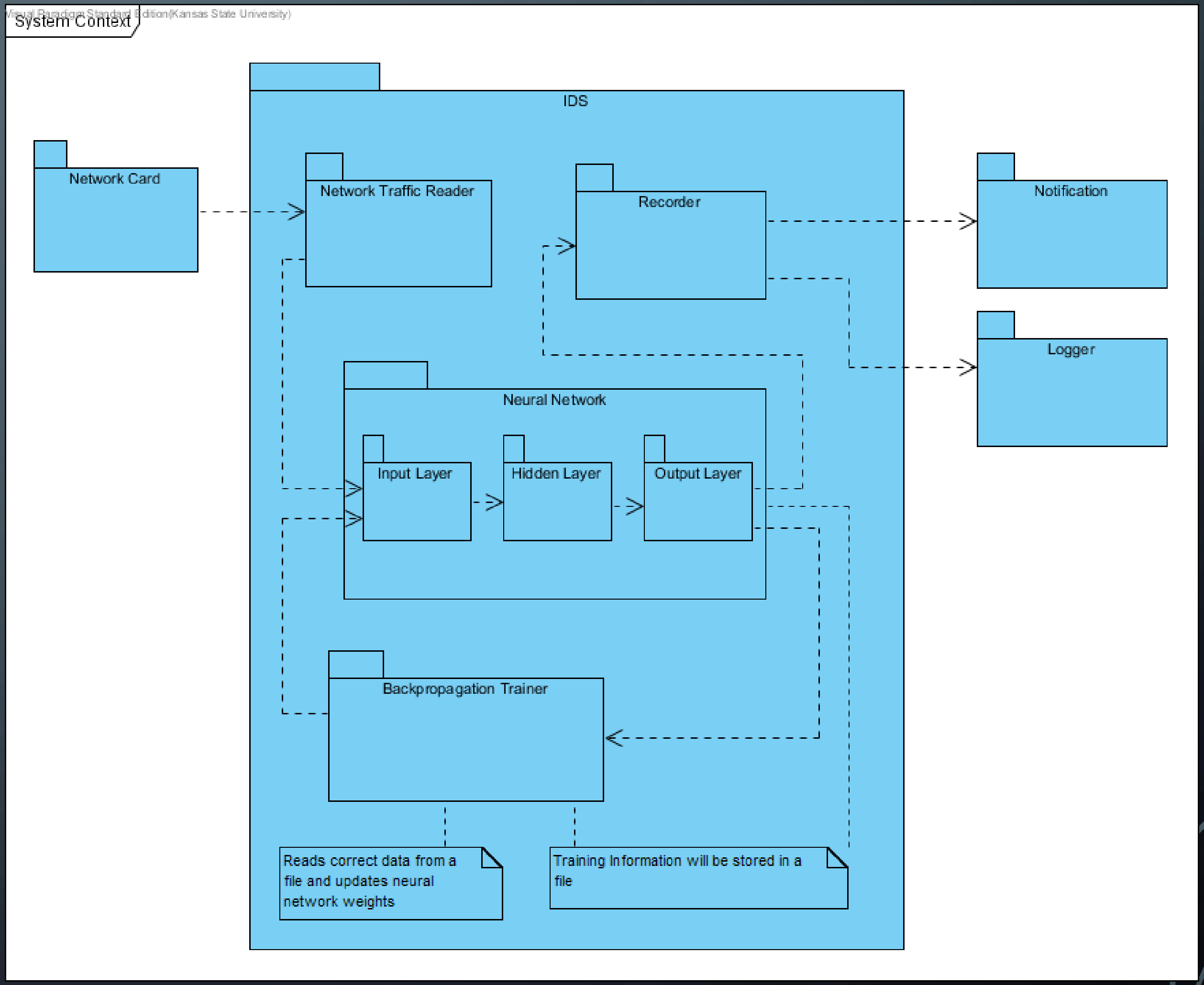
ARCHITECTURE DESIGN

- Architecture is fairly simple from high level
- Diagram shows interactions
- Three main components
    - Network Traffic Reader
    - Neural Network
        - Input Layer
        - Hidden Layer
        - Output Layer
    - Recorder
- Three external nodes
    - Network Data
    - Notification Output
    - Log Output



ARCHITECTURE DESIGN

- Package Design
- Four main packages:
  - Network Traffic Reader
  - Neural Network
    - Input Layer
    - Hidden Layer
    - Output Layer
  - Backpropogation Trainer
  - Recorder
- Three external nodes
  - Network Data
  - Notification Output
  - Log Output

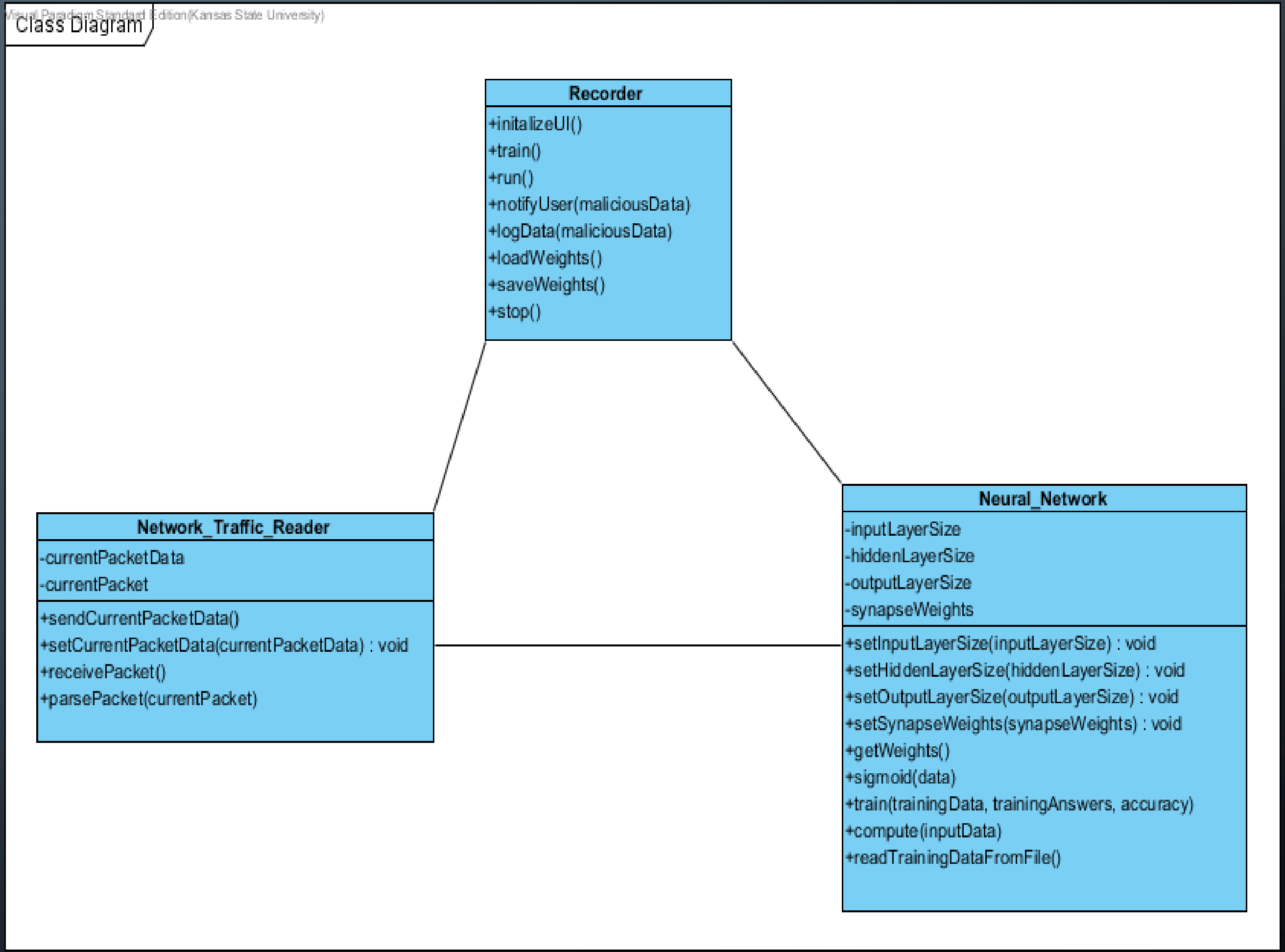


Reads correct data from a file and updates neural network weights

Training Information will be stored in a file

ARCHITECTURE DESIGN

- Class Design Diagram
  - Operations defined in following slides



Class Diagram

Recorder

+initalizeUI()
+train()
+run()
+notifyUser(maliciousData)
+logData(maliciousData)
+loadWeights()
+saveWeights()
+stop()

Network_Traffic_Reader

-currentPacketData
-currentPacket

+sendCurrentPacketData()
+setCurrentPacketData(currentPacketData) : void
+receivePacket()
+parsePacket(currentPacket)

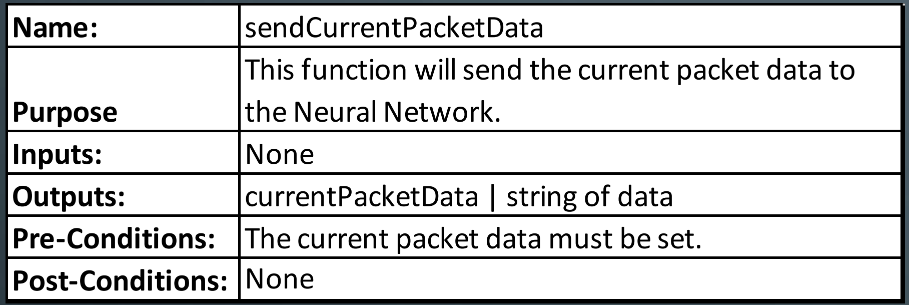
Neural_Network

-inputLayerSize
-hiddenLayerSize
-outputLayerSize
-synapseWeights

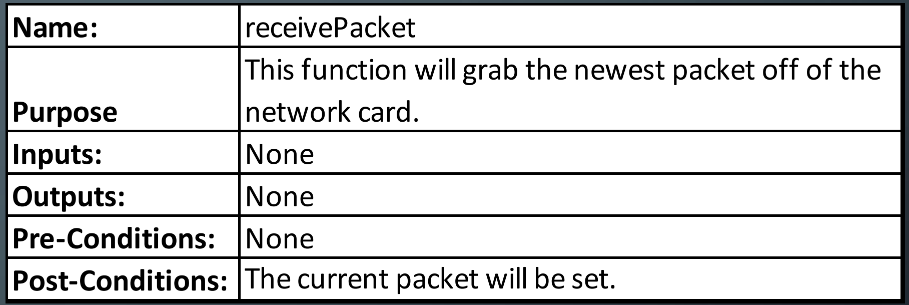
+setInputLayerSize(inputLayerSize) : void
+setHiddenLayerSize(hiddenLayerSize) : void
+setOutputLayerSize(outputLayerSize) : void
+setSynapseWeights(synapseWeights) : void
+getWeights()
+sigmoid(data)
+train(trainingData, trainingAnswers, accuracy)
+compute(inputData)
+readTrainingDataFromFile()

ARCHITECTURE DESIGN

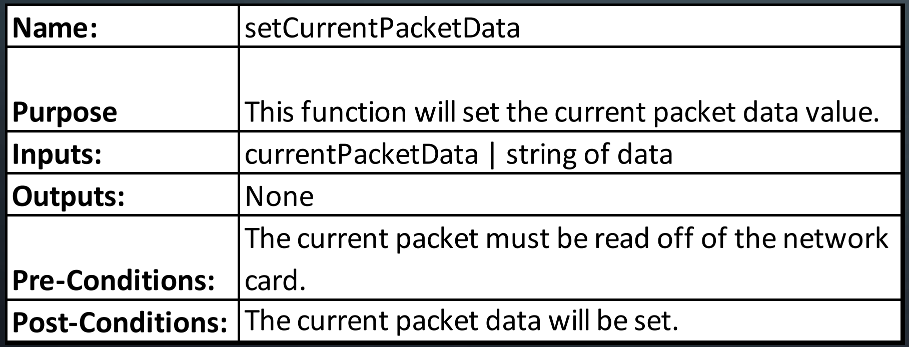
- Network Traffic Reader Class



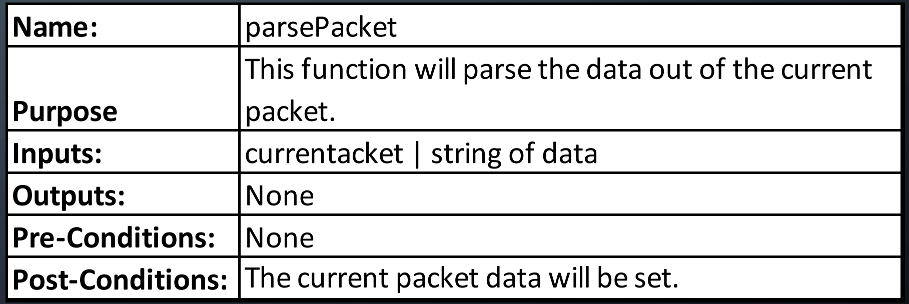
Name:	sendCurrentPacketData
Purpose	This function will send the current packet data to the Neural Network.
Inputs:	None
Outputs:	currentPacketData | string of data
Pre-Conditions:	The current packet data must be set.
Post-Conditions:	None



Name:	receivePacket
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.



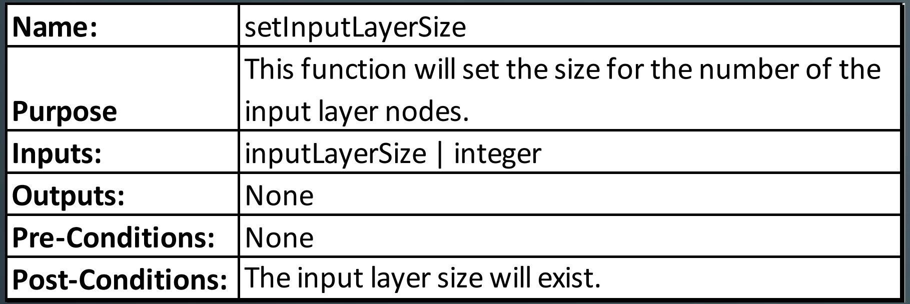
Name:	setCurrentPacketData
Purpose	This function will set the current packet data value.
Inputs:	currentPacketData | string of data
Outputs:	None
Pre-Conditions:	The current packet must be read off of the network card.
Post-Conditions:	The current packet data will be set.



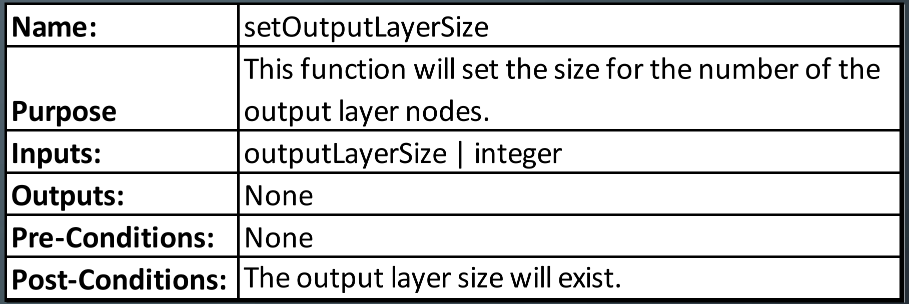
Name:	parsePacket
Purpose	This function will parse the data out of the current packet.
Inputs:	currentacket | string of data
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The current packet data will be set.

ARCHITECTURE DESIGN

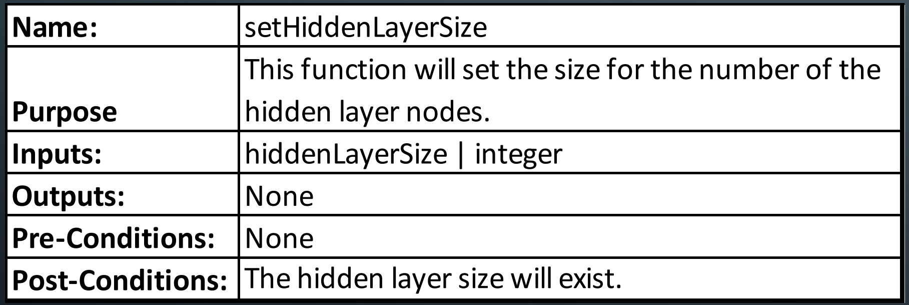
- Neural Network Class



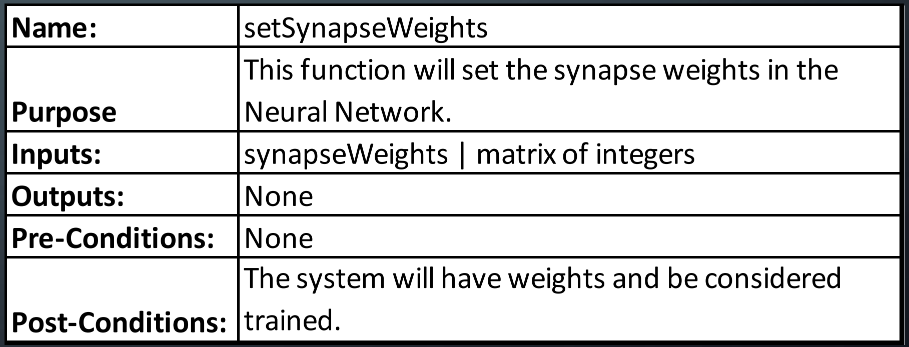
Name:	setInputLayerSize
Purpose	This function will set the size for the number of the input layer nodes.
Inputs:	inputLayerSize | integer
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The input layer size will exist.



Name:	setOutputLayerSize
Purpose	This function will set the size for the number of the output layer nodes.
Inputs:	outputLayerSize | integer
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The output layer size will exist.



Name:	setHiddenLayerSize
Purpose	This function will set the size for the number of the hidden layer nodes.
Inputs:	hiddenLayerSize | integer
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The hidden layer size will exist.



Name:	setSynapseWeights
Purpose	This function will set the synapse weights in the Neural Network.
Inputs:	synapseWeights | matrix of integers
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The system will have weights and be considered trained.

ARCHITECTURE DESIGN

- Neural Network Class

Name:	getWeights
Purpose	This function will get the current synapse weights.
Inputs:	None
Outputs:	synapseWeights | matrix of integers
Pre-Conditions:	The system must have weights currently.
Post-Conditions:	None

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

Name:	readTrainingDataFromFile
Purpose	This function will read and parse all the training data needed from the saved training file.
Inputs:	None
Outputs:	trainingData | matrix of integers trainingAnswers | matrix of integers
Pre-Conditions:	The training data file must exist.
Post-Conditions:	The training data will be read into the Neural Network.

ARCHITECTURE DESIGN

- Recorder Class

Name:	initializeUI
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:	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:	train
Purpose	This function will tell the Neural Network class to begin training using the saved training data.
Inputs:	None
Outputs:	None
Pre-Conditions:	The UI is initialized.
Post-Conditions:	The Neural Network will be trained.

Name:	notifyUser
Purpose	The fuction will notify the user with descriptive string indicating the data that was malicious.
Inputs:	maliciousData | A descriptive string of data
Outputs:	None
Pre-Conditions:	The UI is initialized.
Post-Conditions:	A Notification will pop-up.

ARCHITECTURE DESIGN

- Recorder Class

Name:	logData
Purpose	The fuction will write log data to a log file with descriptive string indicating the data that was malicious.
Inputs:	maliciousData | A descriptive string of data
Outputs:	None
Pre-Conditions:	None
Post-Conditions:	The log file will contain an additional item.

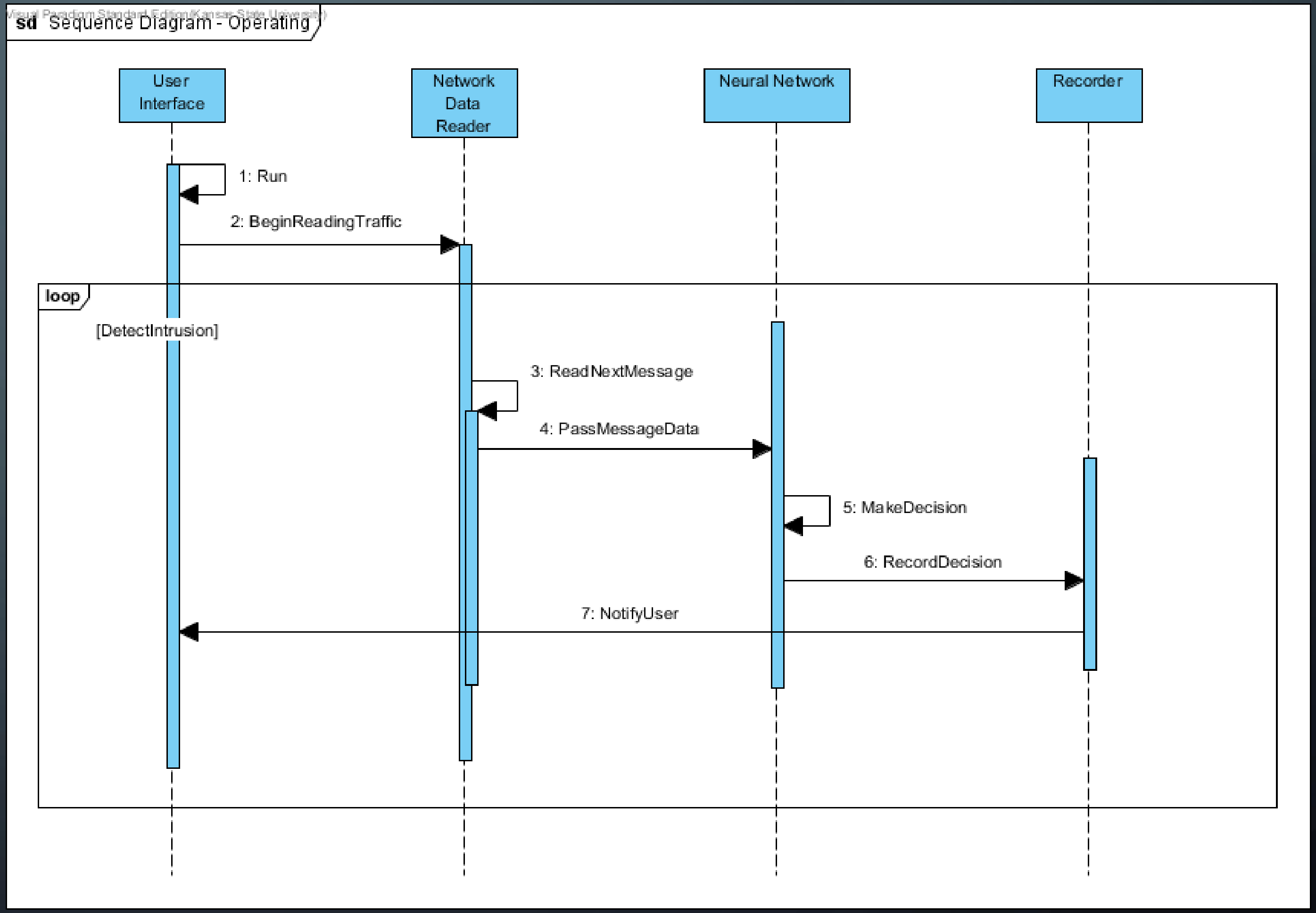
Name:	saveWeights
Purpose	The function will save the current weights to a file.
Inputs:	None
Outputs:	None
Pre-Conditions:	The UI is initialized.
Post-Conditions:	There will be a save weights file.

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

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

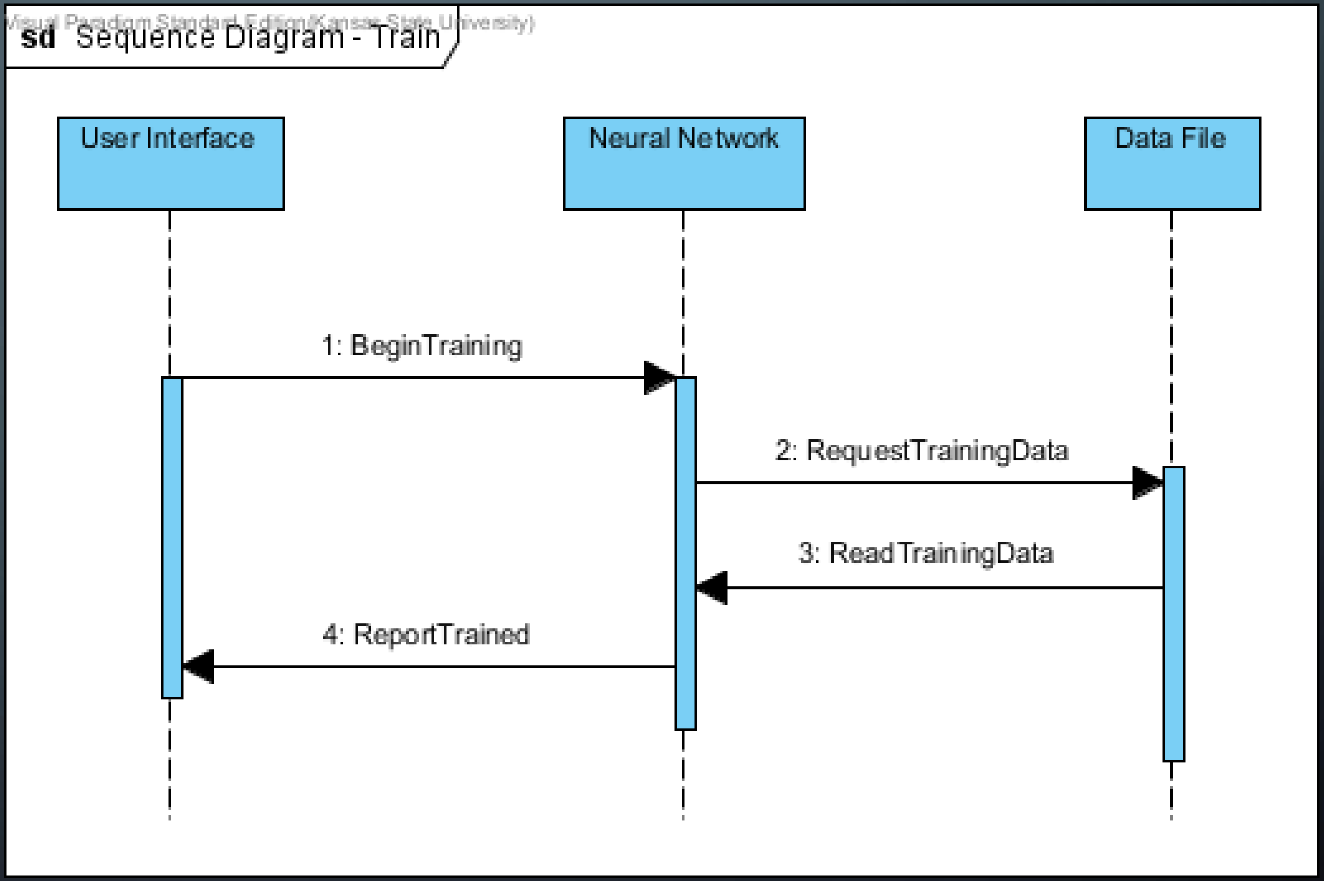
ARCHITECTURE DESIGN

- Operating Sequence Diagram



ARCHITECTURE DESIGN

- Train Sequence Diagram



sd Sequence Diagram - Train

User Interface

Neural Network

Data File

1: BeginTraining

2: RequestTrainingData

3: ReadTrainingData

4: ReportTrained

TEST PLAN

- SR1.1 [Critical Requirement]
  - The system shall be able to read data from the host system network card.
- SR2.1 [Critical Requirement]
  - The system shall be able to interpret the data from the received network traffic and store it in a usable format.
- SR3.1 [Critical Requirement]
  - The system shall be able to determine if the network data received by the host machine is malicious with at least 85% accuracy.
- SR3.2
  - The system shall determine what type of attack is being made to the host network when malicious network traffic is found.
- SR4.1 [Critical Requirement]
  - The system shall be able to train itself through backpropagation on known network traffic data.
- SR5.1 [Critical Requirement]
  - The system shall be able to notify the User of the host system when a malicious attack is encountered.
- SR6.1
  - The system shall be able to log all malicious attacks into a log file.

TEST PLAN

- **Test Case 1: Capturing Data**
  - Requirement(s): SR1.1; SR2.1
  - This test will run the Data Traffic Reader and determine if it can capture the data by capturing the output of this package after it reads the data off the network card. The analysis will look at the format of the data to ensure that the data is formatted correctly.
- **Test Case 2: Correctness Accuracy**
  - Requirement(s): SR3.1
  - This test will run the Neural Network package standalone. The test will consist of feeding the Neural Network with test data, capturing the decisions made, and determining that correctness by comparing the decisions made to the actual results.
- **Test Case 3: Attack Type**
  - Requirement(s): SR3.2
  - This test will run the Neural Network package standalone. The test will consist of feeding the Neural Network with test data, capturing the decisions made, and determining that correctness by comparing the decisions made to the actual results. It will then check to determine if the correct attack type was reported compared to the expected outcome.
- **Test Case 4: Backpropogation Training**
  - Requirement(s): SR4.1
  - This test will run the Neural Network package standalone. The test will consist of training the Neural Network with the saved training data. If the Neural Network can achieve a correctness similar to Test Case 2 then it will pass the test.

TEST PLAN

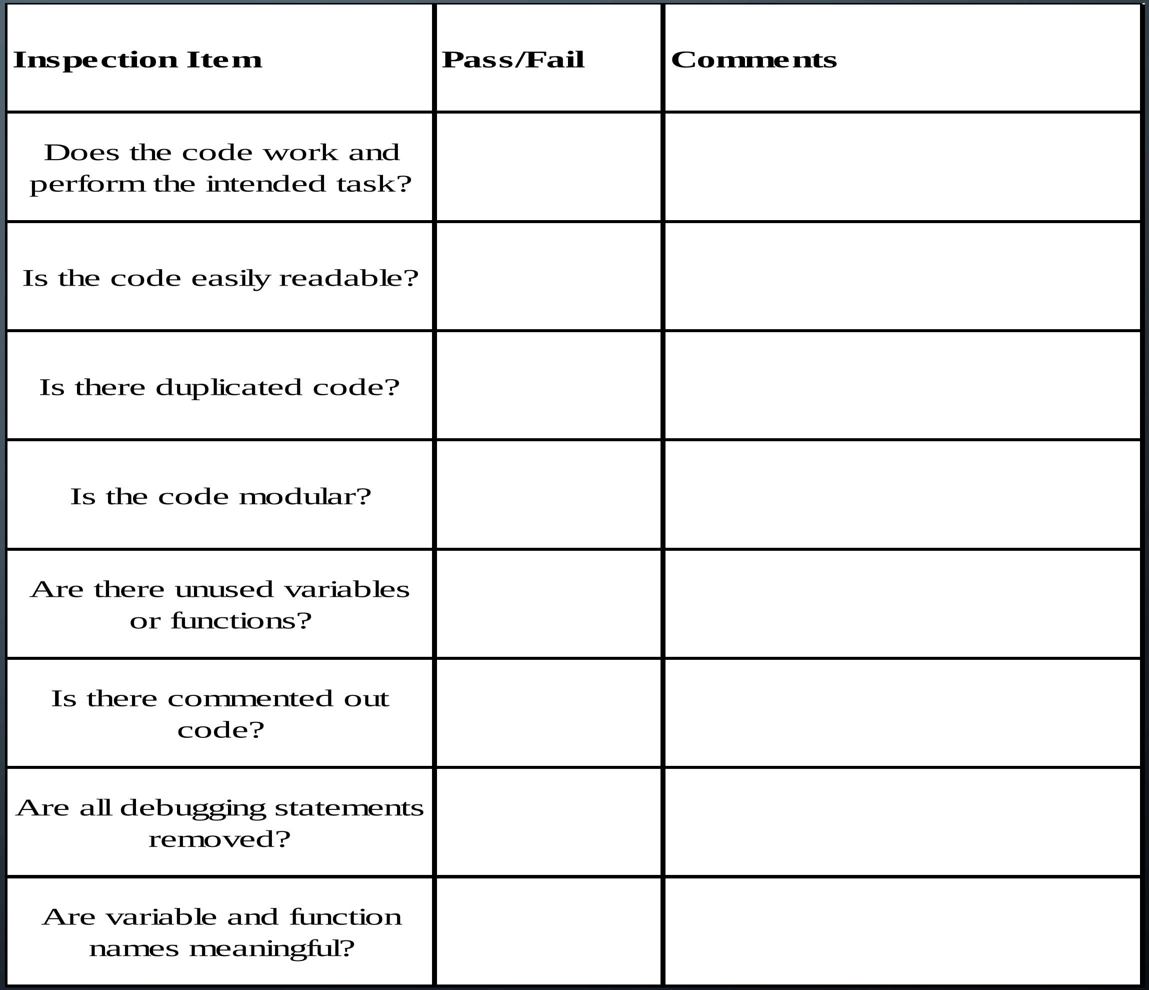
- **Test Case 5: Notification**
  - Requirement(s): SR5.1
  - This test will run the Recorder Package standalone. It will test that it creates a notification when the package receives an input notifying it of a malicious data packet. The notification must have some level of detail about the packet.
- **Test Case 6: Logging**
  - Requirement(s): SR6.1
  - This test will be similar to Test Case 5 but will check a log file instead of a notification.
- **Test Case 7: System Test**
  - Requirement(s): N/A
  - This test will be a full simulation test initializing and running the application. From a separate application I will inject malicious packets to fake malicious data into the system.

TEST PLAN

- Test Deliverables
  - Log file for each test
- Environment Needs
  - Only those needed to run the application

FORMAL TECHNICAL INSPECTION CHECKLIST

- Inspect Prototype Code
- Inspectors:
    - Tracy Marshall
    - Keith Moyer

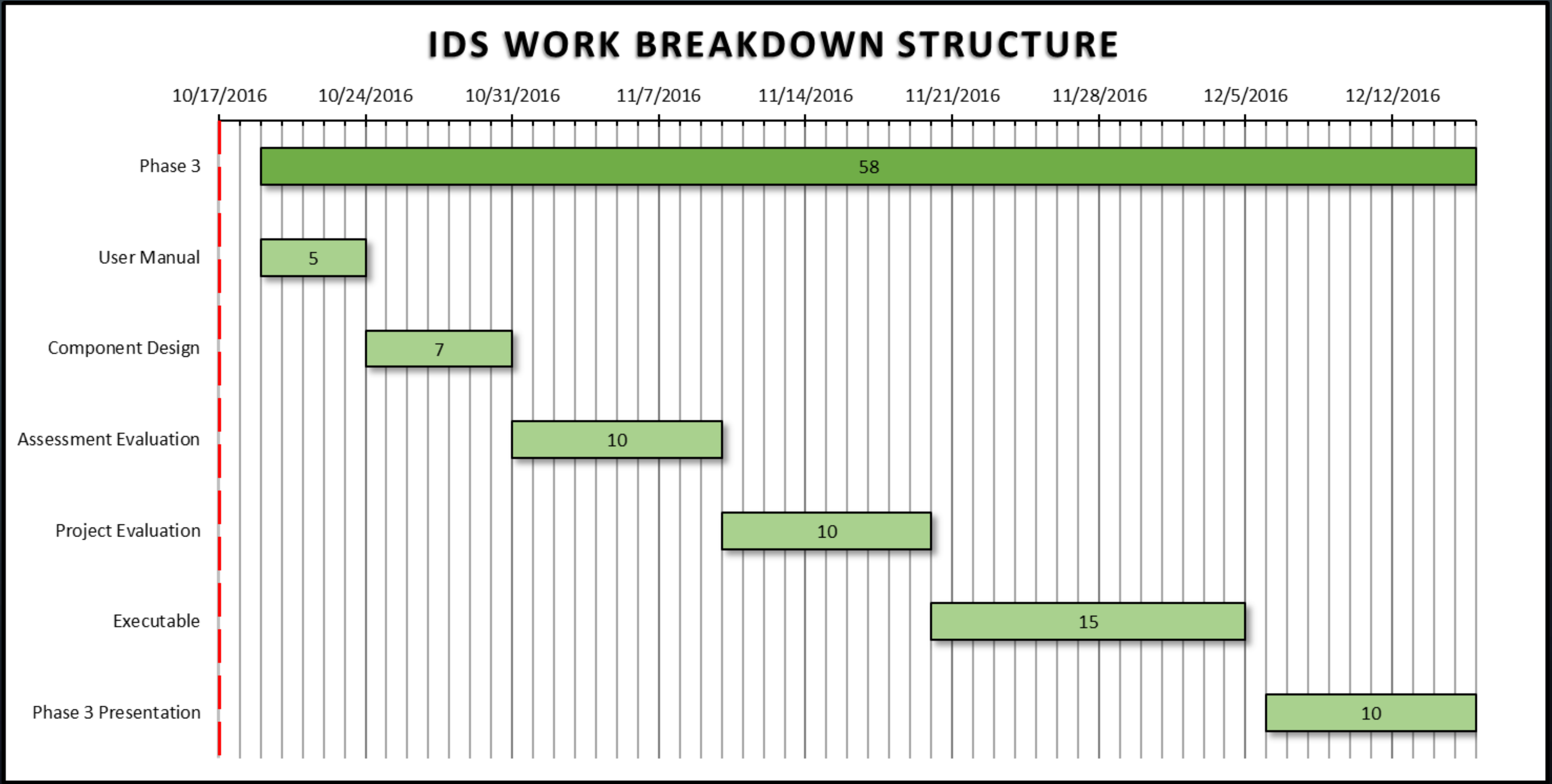


Inspection Item	**Pass/Fail**	**Comments**
Does the code work and perform the intended task?		
Is the code easily readable?		
Is there duplicated code?		
Is the code modular?		
Are there unused variables or functions?		
Is there commented out code?		
Are all debugging statements removed?		
Are variable and function names meaningful?		

RISKS AND CONCERNS

- Loss of data points due to KDD99 expert system data captures
- Achieving the accuracy due to loss of data points

PHASE THREE PLAN



ARCHITECTURE PROTOTYPE DEMONSTRATION

- GitHub Repository Location:
  - https://github.com/bneedy/PyIDS
- All components of the system are working to some degree
- Not fully functional yet – still a prototype
- Can read network traffic off network card and determine decision using neural network

QUESTIONS AND COMMENTS