Centre for Artificial Intelligence and Robotics

Defence Research and Development Organization

Bangalore

Implementation and Evaluation of Network Anomaly Detection Algorithms

Authors:
Mahidhar Chellamani
Nitesh A Jain
Parthasarathy M Alwar
Padmavathi K

Guides:
Dr. Dipti Deodhare,
Scientist 'G'
Mr. Shailesh Sonone,
Scientist 'E'

2014

ABSTRACT

Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behavior. These non-conforming patterns are often referred to as anomalies or outliers. Intrusion detection in computer networks is the process of monitoring for and identifying attempted unauthorized system access or manipulation. This project examines the application of Statistical Modeling for detecting intrusions in computer networks.

Currently, a rule based Network Intrusion Detection System (NIDS) relies on the details of previously known attacks. By knowing the attack vector, rules can be implemented in the firewall and/or other defense tools to recognize the attack and appropriately counter it. However the main drawbacks of this system are - New unknown attacks cannot be countered effectively till encountered and Damage to the system rendering any future counters to the attack useless.

We propose a small NIDS, which is a method to detect and identify attacks. It will concentrate on a limited category of attacks from standard datasets containing well established and tested attack vectors. The network will be modeled by the NIDS using machine learning algorithms based on Naive Bayes Models.

ACKNOWLEDGEMENT

We are happy to present this project after completing it successfully. This project would not have been possible without the guidance, assistance and suggestions of many individuals. We would like to express our deep sense of gratitude and indebtedness to each person who has helped us make this project a success.

We heartily thank our **Principal, Dr. R V Ranganath, BMS Institute of Technology** for his constant encouragement and inspiration in taking up this project.

We are grateful to our **Head of Department, Dr. Thippeswamy G, Dept. of Computer Science and Engineering, BMS Institute of Technology** for his constant encouragement and inspiration in taking up this project.

We sincerely thank our Project guide**, Mrs. A Mari Kirthima, Asst. Professor, Dept. of Computer Science and Engineering,** for her encouragement and advice throughout the course of the Project work.

We gracefully thank our Project Coordinator, **Mr. Rajesh N V, Asst. Professor, Dept. of Computer Science and Engineering,** for his encouragement and advice throughout the course of the Project work.

Our sincere thanks to **Dr. Dipti Deodhare, Scientist 'G', CAIR (DRDO)** and **Mr. Shailesh Sonone, Scientist 'E', CAIR (DRDO)** for providing us the opportunity to do this project under **Defense Research & Development Organization.**

Special thanks to all the staff members of Computer Science Department for their help and kind co-operation.

Lastly we thank our parents and friends for their encouragement and support given to us in order to finish this precious work.

Mahidhar C

Nitesh A Jain

Padmavathi K

Parthasarathy M Alwar

CONTENTS

Chapter 1

INTRODUCTION

1.1 Overview

The current network security technologies (such as Firewall, IDS/ IPS) do provide security to networks but have limitations in terms of detecting only the known attack patterns. The system becomes prone to unknown vulnerabilities until the signatures of such attacks are found and configured in the signature databases of network security solutions. The moment attack pattern changes, the signature based solutions fail.

1.2 Network Security

Network security[1] consists of the provisions and policies adopted by a network administrator to prevent and monitor unauthorized access, misuse, modification, or denial of a computer network and network-accessible resources. Network security involves the authorization of access to data in a network, which is controlled by the network administrator.

1.2.1 Intrusion Detection Systems

An Intrusion Detection System (IDS)[2] is a device or software application that monitors network or system activities for malicious activities or policy violations and produces reports to a management station. Intrusion detection and prevention systems (IDPS) are primarily focused on identifying possible incidents, logging information about them, and reporting attempts.

In a passive system, the intrusion detection system (IDS) sensor detects a potential security breach, logs the information and signals an alert on the console and/or owner. In a reactive system, also known as an intrusion prevention system (IPS), the IPS auto-responds to the suspicious activity by resetting the connection or by reprogramming the firewall to block network traffic from the suspected malicious source. The term IDPS is commonly used where this can happen automatically or at the command of an operator; systems that both “detect (alert)”and “prevent”

1.2.2 Attack Classification

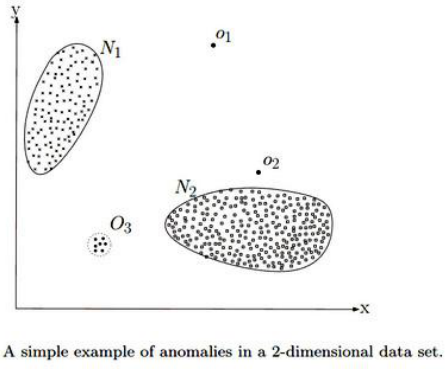
In computer networks, an attack is any attempt to destroy, expose, alter, disable, steal or gain unauthorized access to or make unauthorized use of an asset.[3]

Attack Class	Attack Type
Probe	portsweep, ipsweep, quesosatan, msscan, ntinfoscan, lsdomain.
DoS	apache2, smurf, neptune, tosnuke, land, pod, back, teardrop, tcpreset, syslogd, crashiis, arppoison, mailbomb, selfping, processtable.
R2L	dict, netcat, sendmail, imap, ncftp, xlock, xsnoop, sshtrojan, framespoof, ppmacro, guest, netbus, snmpget, ftpwrite, hhtptunnel, phf, named.
U2R	sechole, xterm, eject, ps, nukepw, secret, perl, yaga, fdformat, ffbconfig, casesen, ntfsdos, ppmacro, loadmodule, sqlattack.

Table 1.1 Attacks present in DARPA 1999 Dataset

1.3 Machine Learning and its Applications

Machine learning, a branch of artificial intelligence, concerns the construction and study of systems that can learn from data. For example, a machine learning system could be trained on network traffic to learn to distinguish between anomalous and normal traffic. After learning, it can used to classify the incoming traffic into anomalous and normal traffic.



A simple example of anomalies in a 2-dimensional data set.

Figure 1.1 Example of Anomalies

Machine Learning is used in a variety of fields, a few of which are mentioned below.

1. **Network Anomaly Detection**: Statistical machine learning is one class of statistical approaches used in the field of Network Anomaly Detection. A choice of a learning scenario depends on the information available in measurements. For example, a frequent scenario is that there are only raw network measurements available and thus unsupervised learning methods are used. If additional information is available, e.g. from network operators, known anomalies or normal network behaviors, learning can be done with supervision. A choice of a mapping depends on the availability of a model, the amount and the type of measurements, and complexity of learning algorithms[4].
2. **Evaluating Learned Knowledge**: Rules induced from training data are not necessarily of high quality. The performance of knowledge acquired in this way is an empirical question that must be answered before that knowledge can be used on a regular basis. One standard approach to evaluation involves dividing the data into two sets, training on the first set, and testing the induced knowledge on the second. One can repeat this process a number of times with different splits, and then average the results to estimate the rules performance on completely new problems. Kibler and Langley (1988) experimental methods of this sort for a broad class of learning algorithms. An important part of the evaluation process is experts' examination of the learned knowledge. Evans and Fisher (1994) encourage an iterative process in developing a fielded application.[5]

Chapter 2

SYSTEM ANALYSIS

2.1 Requirements:

The different hardware and software requirements are covered in the next sub-sections. In the subsequent section, we cover the various functional requirements.

2.1.1 Hardware Requirements:

- i386 or x64 based Processor
- 50 GB Secondary storage
- 8 GB of RAM

2.1.2 Software Requirements:

- GNU/ Linux based OS
- Python 2.6 or greater
- Pycharm community edition
- Numpy 1.7 or greater
- Wireshark
- TCPDump
- Matplotlib

2.1.3 Functional Requirements:

Common functional requirements of an Intrusion Detection System are discussed below.

1. The IDS must continuously monitor and report intrusions.
2. The IDS should be modular and configurable as each host and network segment will require their own tests and these tests will need to be continuously upgraded and eventually replaced with new tests.

3. The IDS must operate in a hostile computing environment and exhibit a high degree of fault-tolerance and allow for graceful degradation.
4. The IDS should be adaptive to network topology and configuration changes as computing elements are dynamically added and removed from the network.
5. Anomaly detection systems should have a very low false alarm rate.
6. The IDS should be capable of providing an automated response to suspicious activity.
7. The ability to detect and react to distributed and coordinated attacks.
8. The IDS should be able to work with other Commercial Off-the-Shelf (COTS) security tools, as no vendor toolset is likely to excel in or to provide complete coverage of the detection, diagnosis, and response responsibilities.
9. IDS data often requires additional analysis to assess any damage to the network after an intrusion has been detected.

2.2 Dataset

The IDS is trained and tested with standard dataset DARPA[19]. Each dataset is a collection of network packets grouped to form **tcpdump** files. Of the different types of packets in the tcpdump files, we filter only **TCP** packets. The structure of a TCP packet is shown in figure 2.1.

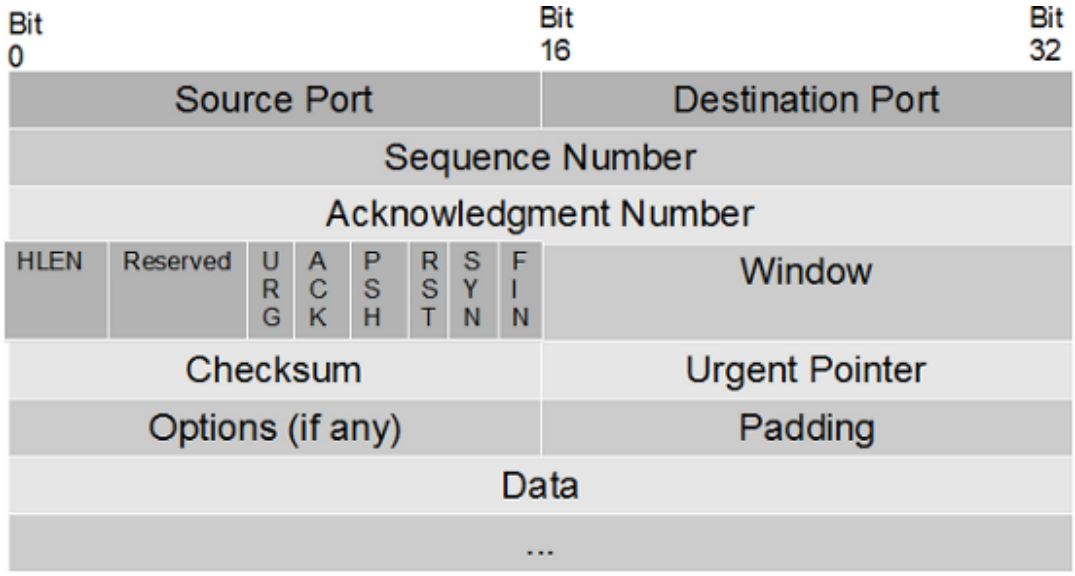


Figure 2.1 TCP Packet Format

Each TCP packet consists of TCP flags like URG, ACK, PSH, RST, SYN and FIN. We filter the flags and group them into six categories as follows:

Type	Flag Set
T1	RST
T2	SYN
T3	ACK
T4	FIN/ACK
T5	PSH/ACK
T6	Others

Table 2.2 TCP Flag Groups

The flags are stored as hexadecimal values which are passed to the program for classifying the incoming traffic as legitimate or malicious. In order to extract only the TCP flags from the datasets we run the following commands -

- ***$tcpdump -r inside/w1mon.tcpdump -w test.pcap***

  This command is used to read the data present in a tcpdump file (w1mon.tcpdump) and write the contents to a pcap file (test.pcap).

- ***$tshark -r test.pcap -Y tcp -T fields -E separator=/s -e ip.dst -e tcp.dstport -e tcp.flags > op_mon***

  This command takes test.pcap file as input, filters the tcp packets, extracts destination IP, destination port and TCP flags and stores the output in op_mon.

- $***cat op_mon | grep "172.16.112.50 23" | cut -d " " -f 3 > op_flag_mon***

  This command selects the particular IP and cuts the third field which contains TCP flag set and stores the output in op_flag_mon.

Finally, we get the TCP flags in the form of Hexadecimal values which are used to train and test the IDS.

The following table consists of information about the number of attack windows and normal windows present in the dataset files considered to train and test the IDS.

		Normal Windows	Attack Windows
DARPA 1	No Attack file	614	0
	Attack file	371	371
DARPA 2	No Attack file	1264	0
	Attack file	582	582
DARPA 3	No Attack file	393	2048
Generated	Attack file	0	11594

Table 2.3 Window statistics of the dataset files

Chapter 3

SYSTEM DESIGN

System design is the process of defining the architecture, modules, components, interfaces and the data for the system to satisfy specified requirements. It pertains to the abstract representation of the data flows, inputs and outputs of the system.

3.1 Design Considerations

In this section, we consider the different modules under consideration, use-case diagram, sequence diagram, data flow diagram and state diagram.

3.1.1 Modules

The system is comprised of five modules. They are -

- Control Unit (CU)
- Traffic Capture Module (TCM)
- Learning Module (LM)
- Detection Module (DM)
- Mitigation Module (MM)

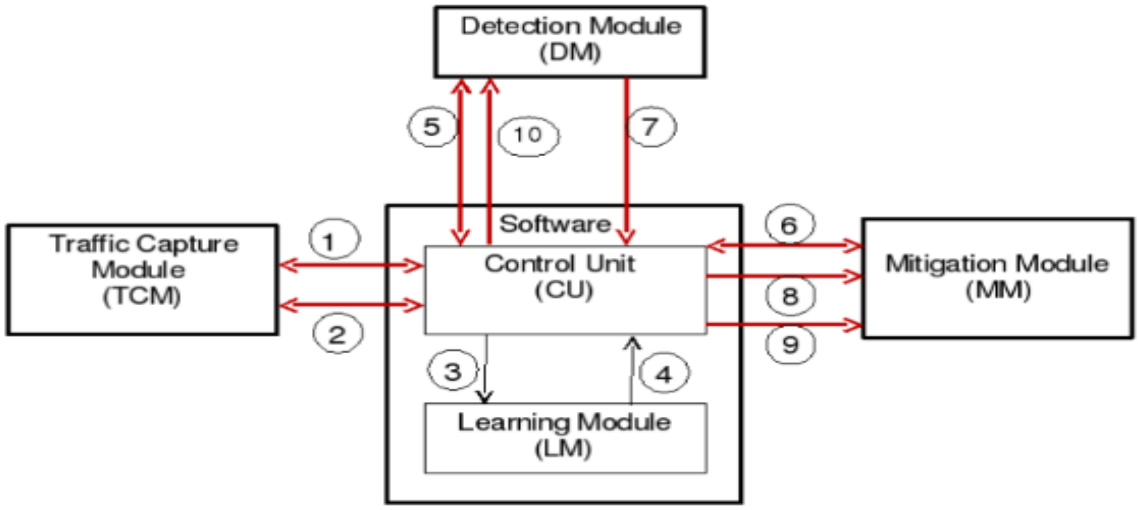


Figure 3.1 Overall System Design

Control Unit (CU):

The Control Unit is a software module, which centrally manages communication between the rest of the modules. CU is resident on the computer, hosting the Learning Module (LM), and interacts with the user (through a Graphical User Interface) to obtain inputs for the above modules. The CU also is responsible for information exchange and dissemination between the modules, and establishes a communication network between each of these modules for this purpose. The CU runs different protocols with the various modules for the task. This is a software module.

Traffic Capture Module (TCM):

The Traffic Capture Module (TCM) is a hardware module, which captures traffic required for learning phase, and computes traffic statistics as required for the Learning Module. The TCM communicates with the Control Unit (CU) to obtain it's inputs, namely stream information. Based on triggers, the values of stream configuration are loaded (by CU), traffic captured and statistics supplied back to the Control Unit.

Learning Module (LM):

The Learning Module (LM) is a software module, which resides in the host computer which is connected to the rest of the hardware modules. The LM accepts configuration details of the server targets, and learning traffic statistics (as captured by the TCM) from the CU, and arrives at model probabilities which will later be used by the Detection Module (DM) for classification of normal traffic/anomalous traffic. It is into this module that the ideas developed as a result of this research work are implemented.

Detection Module (DM):

The Detection Module (DM) is a hardware module, which on input the set of model probabilities of normal traffic per stream (output by LM, and supplied by CU), determines the state of the network (normal or abnormal) in real-time. This module is the operations module for the system. The DM talks with the CU to obtain it's inputs and interrupts the CU in case an attack flag is asserted. This will cause further action by the CU on the Mitigation Module (MM).

Mitigation Module (MM):

The Mitigation Module is a hardware module, which allows input traffic unaltered when attack flag is not set in any stream, and when attack flag is set in one or more streams, it filters traffic for the stream (s) on the basis of a white list of source IPs corresponding to the stream.

Chapter 4

IMPLEMENTATION

This chapter discusses the various implementation details like information about the programming language used, libraries incorporated and the various procedures involved in the operation of this project.

4.1 Language Used for Implementation

The entire system of the project has been written in Python.

4.1.1 Python

Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C. The language provides constructs intended to enable clear programs on both a small and large scale.

Python supports object-oriented, imperative and functional programming. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library. Like other dynamic languages, Python is often used as a scripting language, but is also used in a wide range of non-scripting contexts. Using third-party tools, such as Py2exe or Pyinstaller, Python code can be packaged into standalone executable programs. Python interpreters are available for many operating systems.

Python is a multi-paradigm programming language: object-oriented programming and structured programming are fully supported, and there are a number of language features which support functional programming and aspect-oriented programming (including by meta-programming and by magic methods). Many other paradigms are supported using extensions, including design by contract and logic programming.

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. An important feature of Python is dynamic name resolution (late binding), which binds method and variable names during program execution. The core philosophy of the language is summarized by

the document "PEP 20 (The Zen of Python)", which includes aphorisms such as:

1. Beautiful is better than ugly
2. Explicit is better than implicit
3. Simple is better than complex
4. Complex is better than complicated
5. Readability counts

Libraries like NumPy, SciPy and Matplotlib allow Python to be used effectively in scientific computing, with specialized libraries such as BioPython and Astropy providing domain-specific functionality.

4.2 Libraries Used for Implementation

The project uses the following libraries, which extend the functionality of Python's standard library.

4.2.1 NumPy

Arrays in Python do not have the same definition as that of in traditional programming languages like C. In C arrays are used to store data having the same data type. Python arrays also called lists are similar to C arrays except that they can be used to store mixed type of data. NumPy is an extension to the Python programming language, adding support for large, multidimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode compiler/interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy seeks to address this problem by providing multidimensional arrays and functions and operators that operate efficiently on arrays. Thus any algorithm that can be expressed primarily as operations on arrays and matrices can run almost as quickly as the equivalent C code.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink whereas NumPy is intrinsically

integrated with Python, a more modern, complete, and open source programming language. Moreover complementary Python packages are available like SciPy which is a library that adds more MATLAB-like functionality and matplotlib is a plotting package that provides MATLAB-like plotting functionality.

4.2.2 Matplotlib

Matplotlib is a plotting library for the Python programming language and its NumPy numerical mathematics extension. It provides an object-oriented API for embedding plots into applications. The pylab interface makes matplotlib easy to learn for experienced MATLAB users, making it a viable alternative to MATLAB as a teaching tool for numerical mathematics and signal processing.

Some of the advantages of the combination of Python, NumPy and matplotlib over MATLAB include:

1. Based on Python, a full-featured modern object-oriented programming language suitable for large-scale software development
2. Free and open source
3. Native SVG support

4.3 Procedure Description

The procedure description for various algorithms are mentioned in this section.

4.3.1 Determine Optimal Bands Procedure

This procedure groups the events into bands. The band formation is based on the clustering algorithm.

Input: Event array A, window size for stream N, number of bands K.

Output: 2-dimensional array band, containing indices of per group elements in A.

4.3.2 Update Probabilities Procedure

This procedure computes and updates the probability based on the events.

Input: TCP Stream S, 2-D array band containing band indices of groups of A, number of bands K, Index of observable type L

Output: Probabilities of all events updated into corresponding probability array W index in stream S

4.3.3 Train Procedure

This procedure trains the Naive Bayes Classifier using the TCP flags retrieved from the dataset.

Input: Stream S, Flag Array TF, Window size for stream N , number of bands K.

Output: Updated Stream S with learnt probabilities.

4.3.4 Train Phase Procedure

This procedure initiates training. It calls the training module that trains the NBC against the training dataset. It also calls the module that determines the threshold probability.

Input: Window size N, number of bands K, group count, error proportion t and the filename containing the dataset.

Output: Updated Stream S with learnt probabilities and threshold probability for stream.

4.3.5 Determine Probability Procedure

Probability computation involves computing statistics of a window and determining the probability with which the window would have occurred in the training phase.

Input: Stream with NB probabilities S, packet window W

Output: Probability P of window W

4.3.6 Determine Threshold Probability Procedure

This module computes the threshold probability which will be used to classify if a particular window is either an attack window or not

Input: Stream S, Window size for stream N , group count , flag array

Output: Threshold probability Pt for S

4.3.7 Deploy Procedure

This module tests the input traffic and computes if a window is an attack window or not.

Input: Stream S, filename, Abnormal Window Count (AWC)

Output: State of the system and the window

4.3.8 Parse Procedure

This module reads from the input file. It also groups packets into the 6 groups and plots the packet count graph

Input: Name of the file that contains the flag information

Output: Flag array

Chapter 5

RESULTS

5.1 Results Obtained

The Naive Bayes classifier was tested against the DARPA dataset. The formula to calculate the accuracy is as follows -

$$Accuracy = Detected\ Attacks / Actual\ Attacks$$

Each dataset had two types of traffic – **Clean** traffic and **Mixed** traffic. The clean traffic has no attacks whereas the mixed traffic has both clean and attack packets. Using the above formula the accuracies for each dataset are as follows:

t = 0.091		**Actual**		**Detected**		**Accuracy %**
		Normal	Attack	Normal	Attack	
DARPA 1	No Attack	614	0	607	7	98.86
	Mixed	371	371	371	371	100
DARPA 2	No Attack	1264	0	1210	54	95.73
	Mixed	582	582	550	614	94.50
DARPA 3	Mixed	393	2048	347	2094	88.30
Generated	Attack	0	11594	2	11592	99.98

Table 5.1 Results with t = 0.091

t = 0.08		Actual		Detected		Accuracy %
		Normal	Attack	Normal	Attack	
DARPA 1	No Attack	614	0	614	0	100
	Mixed	371	371	377	365	98.38
DARPA 2	No Attack	1264	0	1241	23	98.18
	Mixed	582	582	582	608	95.53
DARPA 3	Mixed	393	2048	656	1785	87.16
Generated	Attack	0	11594	6	11588	99.95

Table 5.2 Results with t = 0.08

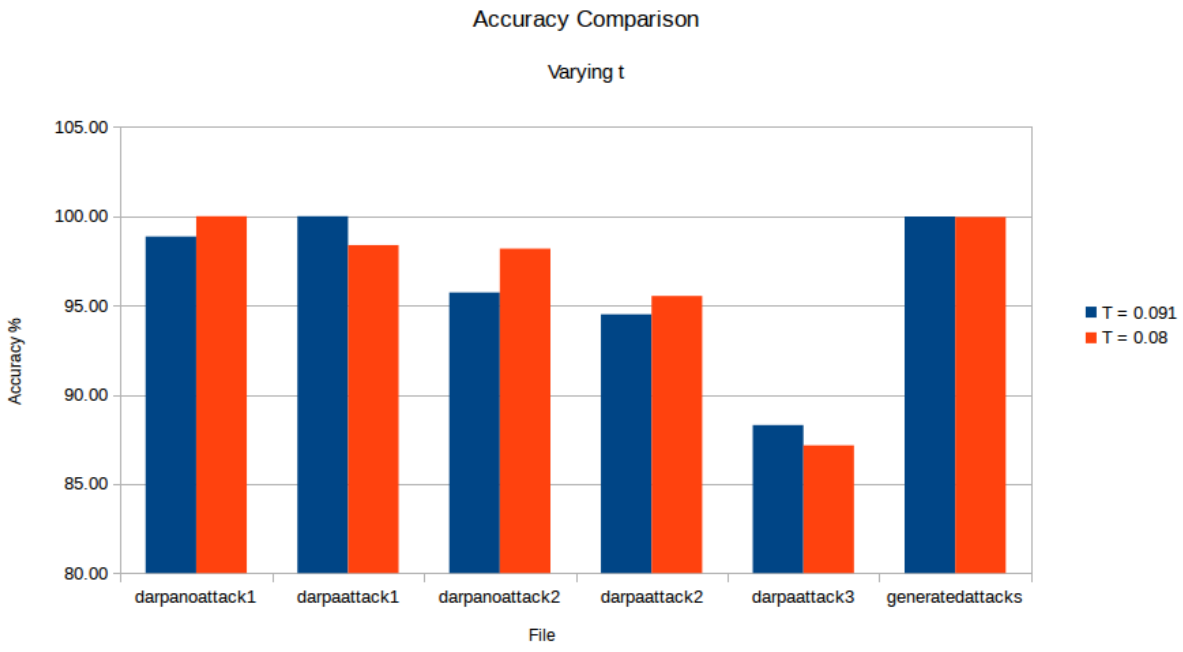
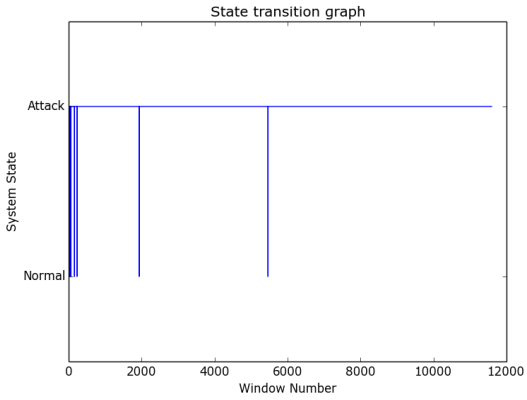
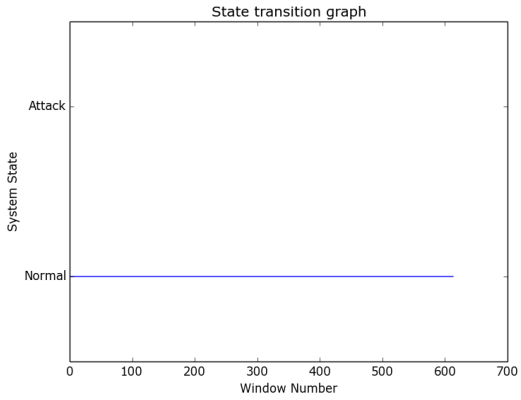


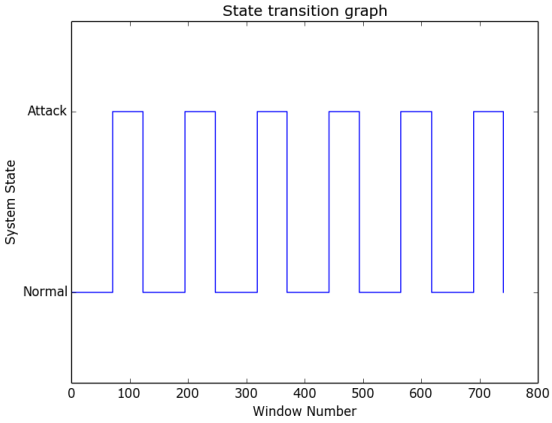
Figure 5.1 Accuracy Comparison



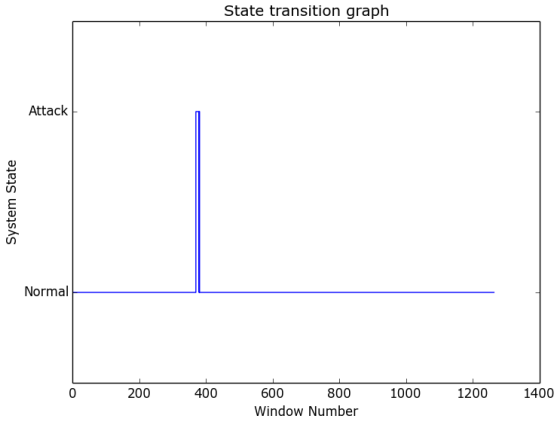
generatedattack output



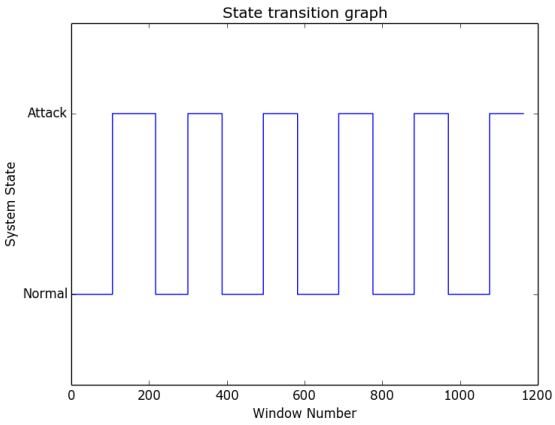
darpanoattack1 output



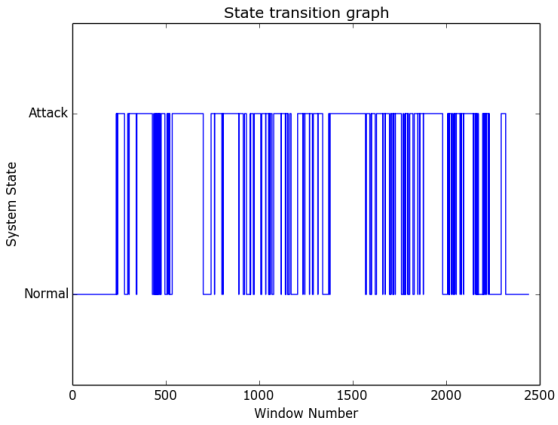
darpaattack1 output



darpanoattack2 output

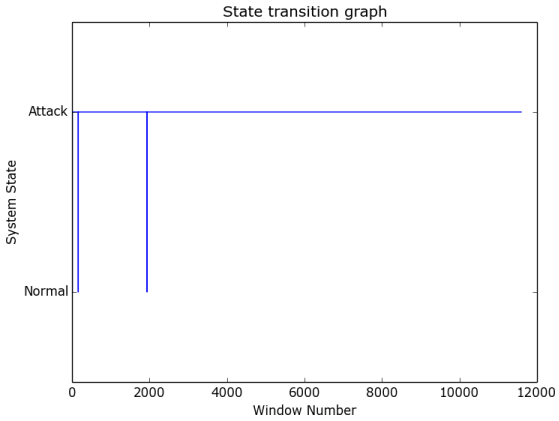


darpaattack2 output

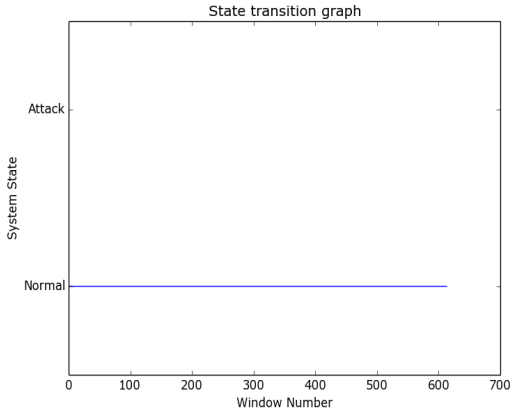


darpaattack3 output

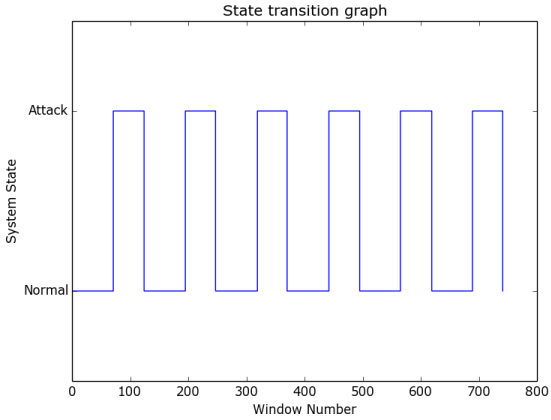
Figure 5.2 System State Graphs for t = 0.08



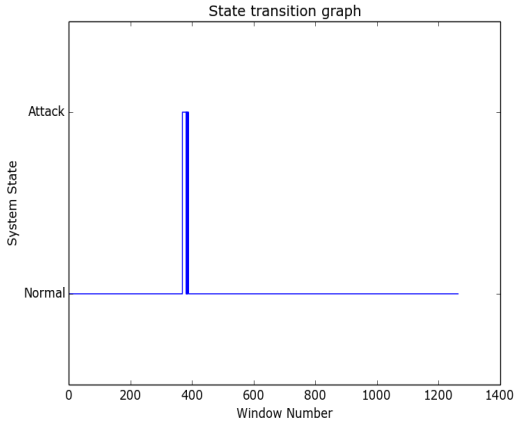
generatedattack



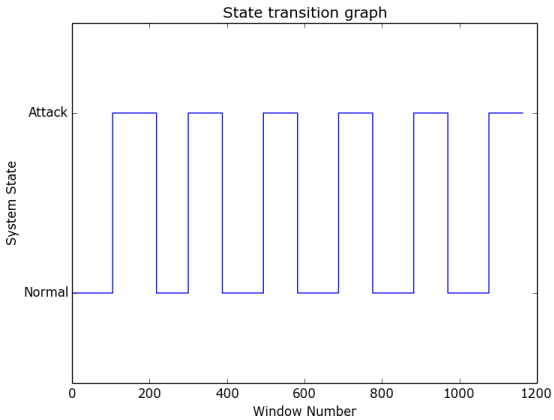
darpanoattack1 output



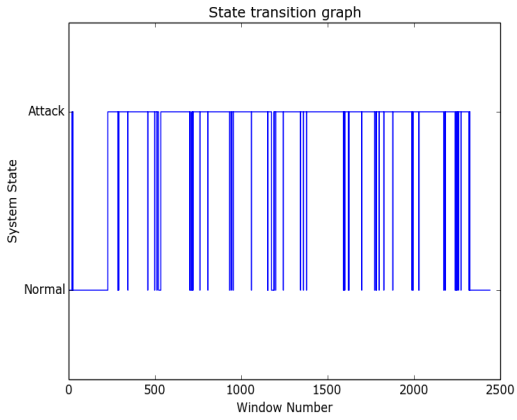
darpaattack1 output



darpanoattack2 output



darpaattack2 output



darpaattack3 output

Figure 5.3 System State Graphs for t = 0.091

Chapter 6

CONCLUSION

The main task of a Naive Bayes Classifier in this project is to detect a Distributed Denial of Service (DDoS) attack in a network. The proposed mechanism has an advantage over signature based systems in that it can easily detect novelty attacks. The proposed system protects the network by constantly monitoring the TCP flags and checking for discrepancies. It classifies windows, that is a group of packets, into either attack or normal depending on the output of the classifier. The Classifier which is implemented in this project is a "site-based", real-time system (with modifications) is simplistic in its design and deployment. Successful testing on generated attacks has resulted in accuracies over 95% which has proved the project to be a success. The proposed mechanism can also be applied for other flag based protocols like the User Datagram Protocol.

The accuracy of the system is calculated based on False Positives (FP) and False Negatives when tested against "No Attack" files and "Attack" files. The accuracies are mentioned below.

No Attack Accuracies (for t=0.091):

	darpanoattack1	**darpanoattack2**
Accuracy	98.86%	95.73%

Attack Accuracies (for t=0.091):

	darpaattack1	**darpaattack2**	**darpaattack3**	**generatedattack**
Accuracy	100%	94.50%	88.30%	99.98%

The proposed system has the following outcomes:

1. Following the author's specifications for testing, the accuracy obtained in this project is coherent with the accuracy obtained by the author.
2. However, the accuracy for DARPA 1998 Week 3 Thursday attack file drastically decreased to 88.30%.

DARPA 1998 Week 3 Thursday Attack File

For the most part of the experiment, we had manually generated the attack file or injected attack flags along with normal flags as specified by the author. The accuracy for these types of files is extremely satisfactory with accuracies ranging over 95%. However, when we tested a pure attack file, obtained from the standard dataset DARPA (Week 3 Thursday), we found a major decrease in the accuracy.

The DARPA 1998 Week 3 Thursday Attack File consists a total of 244,107 packets involving 204,804 attack packets (neptune attack) and 39,303 normal packets. Out of the 204,804 attack packets, the NBC classified 209,400 as attack, which gives us total False Positives of 4600 when compared with the attack windows. We speculate that this increase in false positives due to lack of broad training data. Since the DARPA 1998 Week 3 Thursday Attack file consists of real-world attack, it also consists of normal packets to a large extent along with the attack packets. It can be concluded that the NBC needs a broader training set to handle such type of a realistic attack.

BIBLIOGRAPHY

[1] Simmonds, A; Sandilands, P; van Ekert, L (2004), "*An Ontology for Network Security Attacks*", Lecture Notes in Computer Science 3285: 317–323.

[2] Scarfone, Karen; Mell, Peter (February 2007), "*Guide to Intrusion Detection and Prevention Systems (IDPS)*", Computer Security Resource Center (National Institute of Standards and Technology) (800–94), Retrieved 1 January 2010.

[3] ISO/IEC 27000:2009 from ISO, via their ITTF web site.

[4] Marina Thottan, Guanglei Liu, Chuanyi Ji. "*Anomaly Detection Approaches for Communication Networks*"

[5] Langely, Pat; Simon, Herbert; "*Applications of Machine Learning and Rule Induction*" 19950328164 Institute for the Study of Learning and Expertise Technical Report 95-1 February 15, 1995

[6]DARPA Dataset, 1999, *http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/1999data.html*