IMPROVING MISUSE DETECTION WITH NEURAL NETWORKS

A Thesis Submitted to
The Graduate School of Engineering and Sciences of
İzmir Institute of Technology
In Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in Computer Software

by
Sadettin DEMİRAY

October 2005
İZMİR

We approve the thesis of **Sadettin DEMİRAY**

Date of Signature

……. **17 October 2005**
Asst. Prof. Dr. Tuğkan TUĞLULAR
Supervisor
Department of Computer Engineering
İzmir Institute of Technology

……. **17 October 2005**
Assoc. Prof. Dr. Ahmet KOLTUKSUZ
Department of Computer Engineering
İzmir Institute of Technology

……. **17 October 2005**
Prof. Dr. Şaban EREN
Department of Computer Engineering
Ege University

……. **17 October 2005**
Prof. Dr. Kayhan ERCİYEŞ
Head of Department
Department of Computer Engineering
İzmir Institute of Technology

…….
Assoc. Prof. Dr. Semahat ÖZDEMİR
Head of the Graduate School

ACKNOWLEDGEMENTS

I would like to express sincere gratitude to my thesis supervisor Asst. Prof. Dr. Tuğkan TUĞLULAR for his encouragement, guidance and support throught the development of this thesis.

I would also like to thank my colleagues and friends, Mustafa Özgür TUTUM and Burak Galip ASLAN for their inspiring ideas that enlightened my studies.

Finally, I also owe my special thanks to my family for their encouragements and patience.

ABSTRACT

Misuse Intrusion Detection Systems are rule-based systems that search attack patterns in the data source. Detection ability of misuse detectors is limited to known attack patterns; hence unknown attacks may be missed. In addition, writing new signatures for novel attacks can be troublesome and time consuming. Similarly behavior based IDSs suffered from high rates of false alarms. Artificial neural networks have generalization ability, thus they can be used with intrusion detection system in order to identify normal and attack packets without the need of writing rules. We proposed to use neural networks with network-based IDS. To achieve this, system was trained and tested with both normal and malicious network packets. Backpropagation and Levenberg-Marquardt algorithms were used to train neural networks. For each of these training algorithms a 3-layer and a 4-layer MLP network sets were generated. In addition, self-organizing maps were used to classify attack instances. DARPA 1999 Intrusion Detection Evaluation dataset was used for training and testing, but lack of enough attack patterns in evaluation dataset made us to create a testbed to obtain sufficient malicious traffic. After training was completed, trained neural networks were tested against training dataset and test dataset, which is not part of the training dataset. Results of the experiments showed that, none of the trained backpropagation networks could identify attacks in training and/or testing data sets. But results of the Levenberg-Marquardt networks were more promising as nine of the trained Levenberg-Marquardt networks could identify attack and normal network packets in training and test datasets.

ÖZET

Kötüye kullanım tespit sistemleri, kural tabanlı sistemler olup veri kaynağı içerisindeki saldırı desenlerini arar. Kötüye kullanım tespit sistemlerinin tespit kabiliyetleri sahip oldukları saldırı desenleri ile sınırlıdır. Ek olarak her yeni saldırı için bir saldırı imzası tanımlamak zahmetli ve zaman alıcı bir süreçtir. Benzer olarak davranış tabanlı nüfuz tespit sistemleri de yüksek oranlarda yanlış alarmlarla sorun yaşamaktadır. Yapay sinir ağlarının nüfuz tespit sistemlerinde kullanılmasıyla bir saldırı kuralı tanımlamaya gerek kalmadan normal ve saldırı paketlerinin tanınması sağlanabilir.

Bu çalışma yapay sinir ağlarının nüfuz tespit sistemlerinde kullanılmasını kapsamaktadır. Bu amaca ulaşabilmek için yapay sinir ağları hem saldırı hem de normal ağ paketleriyle eğitilmiştir. Geri yayılım algoritması ve Levenberg-Marquardt eğitim algoritmaları yapay sinir ağlarının eğitilmesinde kullanılmıştır. Her bir eğitim algoritması için biri 3-katmanlı biri de 4-katmanlı olmak üzere iki grup çok düzeyli algılayıcı ağları yaratılmıştır. Çok düzeyli algılayıcı ağlara ek olarak kendi kendini organize eden yapay sinir ağları da saldırıları gruplandırmak için kullanılmıştır. Eğitim ve test için 1999 DARPA Nüfuz Tespit Sistemleri Değerlendirme veri seti kullanılmıştır. Ancak bu veri setindeki saldırı paketlerinin sayısının yapay sinir ağlarını eğitimi için yetersiz kalmasından dolayı, saldırı paketleri bir simülasyon ortamında yaratılmıştır. Eğitilen yapay sinir ağları, eğitim veri seti ve eğitim veri setinin içinde olmayan verilerin bulunduğu bir test veri setiyle test edilmiştir. Deney sonuçlarında geri yayılım algoritması ile eğitilen yapay sinir ağlarının eğitim ve/veya test veri setindeki normal ve saldırı paketlerini ayrıt etmede başarısız kaldığı gözlemlenmiştir. Levenberg-Marquardt algoritması ile eğiten dokuz yapay sinir ağı, eğitim ve test veri setindeki saldırı ve normal paketleri ayırt edebilmeyi başarmıştır.

TABLE OF CONTENTS

LIST OF FIGURES ix

LIST OF TABLES xi

CHAPTER 1. INTRODUCTION 1

CHAPTER 2. BACKGROUND

2.1. Intrusion Detection Systems 6

2.1.1. Information Sources 7

2.1.1.1. Host-based IDSs 7

2.1.1.1.1. Advantages of Host-based IDSs 8

2.1.1.1.2. Disadvantages of Host-based IDSs 8

2.1.1.2. Network-based IDSs 8

2.1.1.2.1. Advantages of Network-based IDSs 9

2.1.1.2.2. Disadvantages of Network-based IDSs 9

2.1.2. IDS Analysis 9

2.1.2.1. Anomaly Detection 10

2.1.2.1.1. Advantages of Anomaly IDSs 10

2.1.2.1.2. Disadvantages of Anomaly IDSs 11

2.1.2.2. Misuse Detection 11

2.1.2.2.1. Advantages of Misuse IDSs 12

2.1.2.2.2. Disadvantages of Misuse IDSs 12

2.1.3. Response Component 12

2.2. Neural Networks 12

2.2.1. Models of Neurons 13

2.2.2. Neural Network Architectures 15

2.2.2.1. Multi-Layer Perceptron 15

2.2.2.2. Self Organizing Maps 17

2.2.3. Learning in Neural Networks 18

2.2.3.1. Backpropagation Algorithm 19

2.2.3.2. Levenberg-Marquardt Algorithm 21
2.3. DARPA 1999 Intrusion Detection Evaluation..... 22

CHAPTER 3. INTRUSION DETECTION WITH NEURAL NETWORK..... 25
3.1. Motivation..... 25
3.2. Earlier Work 26
3.3. Advantages of NN Based IDS 34
3.4. Disadvantages of NN Based IDS..... 34

CHAPTER 4. METHODOLOGY 36
4.1. Compose Training/Testing Data..... 37
4.2. Preprocess Training/Testing Data..... 39
4.3 Neural Network Structure..... 42
4.3.1. MLP Networks..... 42
4.3.2. Self-Organizing Maps..... 43
4.4. Training the Neural Networks 44
4.5. Testing the Neural Networks 45

CHAPTER 5. RESULTS AND DISCUSSION..... 46
5.1. Verification Step 46
5.1.1. 3-Layer Backpropagation Neural Networks 46
5.1.2. 3-Layer Levenberg-Marquardt Neural Networks 49
5.1.3. 4-Layer Backpropagation Neural Networks 51
5.1.4. 4-Layer Levenberg-Marquardt Neural Networks 53
5.1.5. SOMs 55
5.2. Generalization Step..... 56
5.2.1. 3-Layer Backpropagation Neural Networks 56
5.2.2. 3-Layer Levenberg-Marquardt Neural Networks 58
5.2.3. 4-Layer Backpropagation Neural Networks 60
5.2.4. 4-Layer Levenberg-Marquardt Neural Networks 61
5.2.5. SOMs 63
5.3. Analysis 64
5.4. Discussion..... 65

CHAPTER 6. CONCLUSION 66

REFERENCES 68

APPENDICES

APPENDIX A. LIST OF THE SELECTED VULNERABILITIES 70

APPENDIX B. LIST OF THE EXPERIMENTAL RESULTS 81

LIST OF FIGURES

Figure **Page**

Figure 1.1. CERT reported incidents by year. 1
Figure 1.2. Timeline of the evolution of intrusion detection systems. 3
Figure 2.1. Block diagram of a typical anomaly detection system. 10
Figure 2.2. Block diagram of a typical misuse detection system. 11
Figure 2.3. Major components of an artificial neuron. 13
Figure 2.4. Most popular activation functions. 14
Figure 2.5. A MLP network with two hidden layers. 16
Figure 2.6. An example of self-organizing map network. 17
Figure 2.7. Global and local minimum in error surface. 20
Figure 2.8. DARPA 1999 Intrusion Detection Evaluation Testbed. 23
Figure 3.1. Block diagram of enhanced intrusion detection system. 28
Figure 3.2. A diagram of an IDA. 30
Figure 3.3. General schematic for the Network Monitor Stack. 31
Figure 4.1. Steps to be taken in order to build neural network based IDS. 36
Figure 4.2. Simulation environment used to generate malicious traffic. 38
Figure 4.3. Summary of training and testing datasets. 39
Figure 4.4. ER diagram of the database created by Snort. 40
Figure 4.5. A sample training session where performance goal was met in 38 epochs. 45
Figure 5.1. Number of false positives for 3-layer backpropagation networks. 47
Figure 5.2. Number of false negatives for 3-layer backpropagation networks. 48
Figure 5.3. Number of false positives for 3-layer Levenberg-Marquardt networks. 49
Figure 5.4. Number of false negatives for 3-layer Levenberg-Marquardt networks. 50
Figure 5.5. Number of false positives for 4-layer backpropagation networks. 51
Figure 5.6. Number of false negatives for 4-layer backpropagation networks. 52
Figure 5.7. Number of false positives for 4-Layer Levenberg-Marquardt networks. 53
Figure 5.8. Number of false negatives for 4-layer Levenberg-Marquardt networks. 54
Figure 5.9. Number of false positives for 3-layer backpropagation networks. 56
Figure 5.10. Number of false negatives for 3-layer backpropagation networks. 57
Figure 5.11. Number of false positives for 3-layer Levenberg-Marquardt networks. 58

Figure 5.12. Number of false negatives for 3-layer Levenberg-Marquardt networks. ... 59
Figure 5.13. Number of false positives for 4-layer backpropagation networks. 60
Figure 5.14. Number of false negatives for 4-layer backpropagation networks..... 61
Figure 5.15. Number of false positives for 4-layer Levenberg-Marquardt networks. 62
Figure 5.16. Number of false negatives for 4-layer Levenberg-Marquardt networks. ... 63

LIST OF TABLES

Table **Page**

Table 1.1. Economic impact of malicious code. 2
Table 3.1. Summary of Earlier Studies 32
Table 4.1. Sample preprocessed input vector. 41
Table B.1. Number of false positives for 3-layer backpropagation networks. 81
Table B.2. Number of false negatives for 3-layer backpropagation networks. 82
Table B.3. Number of false positives for 3-layer Levenberg-Marquardt networks. 83
Table B.4. Number of false negatives for 3-layer Levenberg-Marquardt networks..... 83
Table B.5. Number of false positives for 4-layer backpropagation networks. 84
Table B.6. Number of false negatives for 4-layer backpropagation networks. 85
Table B.7. Number of false positives for 4-Layer Levenberg-Marquardt networks. 85
Table B.8. Number of false negatives for 4-layer Levenberg-Marquardt networks..... 86
Table B.9. Number of false positives for 3-layer backpropagation networks. 87
Table B.10. Number of false negatives for 3-layer backpropagation networks. 87
Table B.11. Number of false positives for 3-layer Levenberg-Marquardt networks. 88
Table B.12. Number of false negatives for 3-layer Levenberg-Marquardt networks..... 89
Table B.13. Number of false positives for 4-layer backpropagation networks. 89
Table B.14. Number of false negatives for 4-layer backpropagation networks. 90
Table B.15. Number of false positives for 4-layer Levenberg-Marquardt networks. 91
Table B.16. Number of false negatives for 4-layer Levenberg-Marquardt networks..... 91

CHAPTER 1

INTRODUCTION

Automated attacking tools and high availability of these tools on Internet have raised the need of computer security systems. Firewalls and security policies are the first defense lines in order to protect computer systems and prevent attackers to harm computer systems. However, building a complete secure computer system is still a dream. This is due to the fact that, application programs will always contain unknown bugs and vulnerabilities. In addition, attackers continuously find new techniques to exploit vulnerabilities in the computer systems. Hence, despite the security precautions, computer attacks are continuously increasing (attack incidents reported to CERT (WEB_1 2005) by year is given in Figure 1.1).

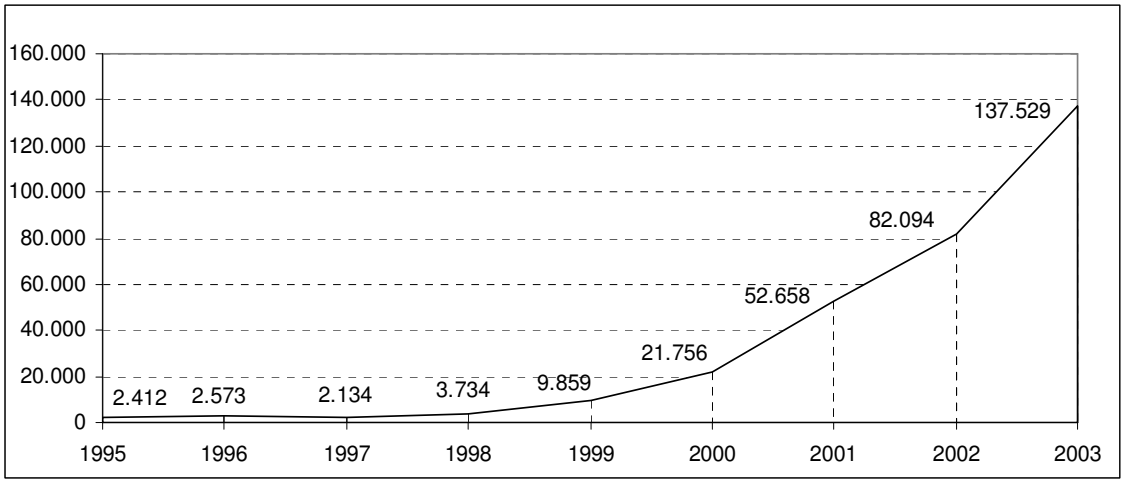

Figure 1.1. CERT reported incidents by year.

Electronic commerce became widespread; hence attacks to the computer systems could result in loss of billions of dollars. Worldwide economic cost of the malicious codes is $0.5 billion in 1995, $1.8 billion in 1996, $3.3 billion in 1997, $6.1 billion in 1998, $12.1 billion in 1999, $17.1 billion in 2000 and $13.2 billion in 2001. Major incidents related to the viruses, worms and other malicious software are given in Table 1.1 (WEB_2 2005).

Table 1.1. Economic impact of malicious code.

Name	Date	Impact
Morris Worm	1988	• Stopped 10% of computers connected to Internet
Melissa Virus	May 1999	• 100,000 computers in one week • Loss of $1.5 billion
Explorer Virus	June 1999	• Loss of $1.1 billion
Love Bug Virus	May 2000	• Loss of $8.75 billion
Sircam Virus	July 2001	• 2.3 million computers infected • Loss of $1.25 billion
Code Red Worm	July 2001	• 359,000 computers infected in less than 14 hours • Loss of $2.75 billion
Nimda Worm	Sept. 2001	• 160,000 computers infected at peak • Loss of $1.5 billion
Klez	2002	• Loss of $750 million
BugBear	2002	• Loss of $500 million
Badtrands	2002	• Loss of $400 million
Sapphire/Slammer Worm	Jan. 2003	• Infected 90% of vulnerable hosts in just 10 minutes • 75,000 hosts infected at peak • Loss of $1.5 billion
Blaster	2003	• Loss of $750 million
Nachi	2003	• Loss of $500 million
SoBig.F	2003	• Loss of $2.5 billion
MyDoom Worm	Jan. 2004	• Fastest spreading mass-mailer worm to date • 100,000 instances of the worm intercepted per hour • Loss of more than $4.0 billion
Witty Worm	March 2004	• First widely propagated worm to carry a destructive payload

A computer system should provide confidentially, integrity and availability against attempts try to exploit vulnerabilities in the operating system and in application programs. Confidentially is the protection of private information from disclosure of unauthorized users. Integrity is the protection of the sensitive data from modification of unauthorized users and availability states that data can be accessible any time by the legitimate users.

If certain countermeasures are taken, most of the intrusions can be prevented, as most of them are results of known vulnerabilities. However, it is still impossible to

secure the complete computer system, as there will be always unknown security breaches. Consequently, it is important to detect intrusions or intrusion attempts in order to take precautions as to prevent similar intrusions. This research field is called intrusion detection.

The notion of intrusion detection was born with Anderson's paper, "Computer Security Threat Monitoring and Surveillance", in the beginning of 1980. In his seminal paper, written for a government organization, Anderson suggested using audit trails to track computer misuses and to understand user behavior. With release of the paper, concepts of detecting misuse and user activities emerged. Since then, many researches have been conducted and intrusion detection systems have been significantly improved. Timeline of the evolution of intrusion detection systems is given in Figure 1.2.

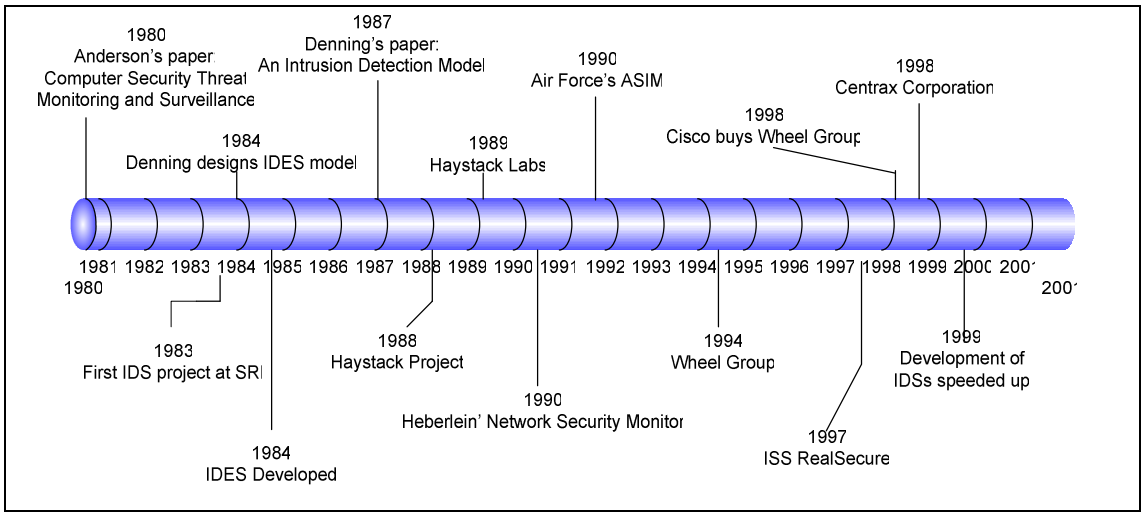

Figure 1.2. Timeline of the evolution of intrusion detection systems.

In 1983, Denning in SRI International conducted a study to create user profiles by analyzing audit trails of the government mainframe computers. One year later the first prototype for intrusion detection, the Intrusion Detection Expert System (IDES) was formed with the help of Denning. IDES analyzes audit trails from government systems and tracks user activity. IDES provided a foundation to the intrusion detection development. In 1987, Denning published a paper, "An Intrusion Detection Model", for the 1986 IEEE Symposium on Security and Privacy and explained how anomalous activity could be used as an indicator of potential security incidents.

The Haystack prototype was developed for the detection of intrusions in a multi-user Air Force computer system at University of California Davis' Lawrence Livermore

Laboratories in 1988. The prototype Intrusion Detection System (IDS) analyzes audit data by comparing it with defined patterns.

In 1989, the developers of Haystack established the commercial company, Haystack Labs, and developed “last” technology intrusion detection system, Stalker. Stalker is a host-based, pattern matching system which can manually and automatically query the audit data.

In 1990, Todd Heberlein, a student at the University of California, introduced the idea of network intrusion detection and built an IDS system called Network Security Monitor (NSM) which was the first network intrusion detection system. NSM listens passively network traffic that passes through a broadcast LAN, and extracts intrusive behavior from this input. NSM can monitor a network of heterogeneous hosts without having to convert different audit trail formats into a standard format.

The Air Force's Cryptologic Support Center developed the Automated Security Measurement System (ASIM) to monitor network traffic on the US Air Force's network. ASIM considerably improved scalability and portability issues that previous network intrusion detection products suffered.

The development group on the ASIM project formed a commercial company in 1994, the Wheel Group. Their product, NetRanger, scans traffic for signature of misuse, provides real-time alarm and gives details of the attacks. However, during these years development of commercial intrusion detection systems slowed down and only speeded up again around 1997.

ISS developed a network intrusion detection system called RealSecure in 1997. A year later, Cisco recognized the importance of network intrusion detection and purchased the Wheel Group. Similarly, the first host-based intrusion detection company, Centrax Corporation, was formed.

In the last few years, the intrusion detection field has grown considerably and therefore a large number of IDSs have been developed. The initial IDSs were anomaly detection tools but today, most of the commercial IDSs are misuse detection tools. IDSs have become a necessity, as number of computer systems connected to networks increased greatly.

The goal of this research is to propose and analyze the applicability of neural networks in the field of intrusion detection. Proposed neural network based intrusion detection system is network-based in the sense that, it uses network data to determine whether an intrusion has taken place.

Remainder of the thesis is organized as follows. Basic concepts of intrusion detection, neural networks and information about DARPA 1999 Intrusion Detection Evaluation are given in chapter two. Employing neural networks in the field of intrusion detection and related works are given in chapter three. Methodology used in this research is described in chapter four. The results that are obtained in the experiments in this research are examined in chapter five.

CHAPTER 2

BACKGROUND

This chapter presents basic technical background of intrusion detection systems, neural networks, 1999 DARPA Intrusion Detection Evaluation and its dataset, which is used to train and test the proposed system.

2.1. Intrusion Detection Systems

Pioneering work in the field of intrusion detection was performed by Anderson in the early 1980s. Anderson defines an intrusion as any unauthorized attempt to access, manipulate, modify, or destroy information, or to render a system unreliable or unusable (Anderson 1980). Intrusions are caused by attackers accessing the systems from Internet, authorized users of the systems who attempt to gain additional privileges for which they are not authorized, and authorized users who misuse the privileges given to them. IDSs are software or hardware products that monitor the system in question and try to detect any attack against the system.

A truly secure system is still a dream, as there are always bugs in application programs, and also communication protocols always have vulnerabilities that can be exploited by attackers. In addition, passwords can be cracked, users can loose their passwords, and entire crypto system can be broken. As a result, security mechanisms (e.g. firewalls), which are deployed to protect the information system, may not be able to prevent all security breaches. IDSs are usually deployed along with the other security mechanisms, such as access control, authentication and firewalls, as a last defense line to improve security of the information system.

The main goal of an IDS is to provide high rates of attack detection with very small rates of false alarms. There are two types of errors that are important to know in intrusion detection:

- False positives: False positives are the errors occurring when IDS flags a normal activity as an attack. Simply, false positives are false alarms.

- False negatives: False negatives are the errors occurring when IDS fails to detect an ongoing attack.

An intrusion detection system consists of three functional components (Bace and Mell 2001), namely:

1. Information source that provides a stream of event records;
2. Analysis engine that finds signs of intrusions; and
3. Response component that generates reactions based on the outcome of the analysis engine.

In the following subsections these three functional components are inspected in detail.

2.1.1. Information Sources

The first component of an IDS is the data source, where input information, which will be analyzed, is collected. Input information can be audit trails, system logs or network packets. According to the data sources used, IDSs can be classified into two categories, host-based IDSs and network-based IDSs.

2.1.1.1. Host-based IDSs

Host-based IDSs monitor activities within an individual computer system and operate on information obtained from these activities. As target environment was mainframe environment, and all users were local to the system, first researches in the field of IDSs were performed on the host-based IDSs.

Host-based IDSs generally use two information sources, operating system audit trails, and system logs. Operating system trails are usually generated at the kernel level; hence they are more detailed and better protected than system logs. However, system logs are simpler and smaller than operating system trails; consequently they can be more easily interpreted.

2.1.1.1.1. Advantages of Host-based IDSs

1. As host-based IDSs monitor local activities, thus they can detect attack that can not be detected by network-based IDSs.
2. Information sources of the host-based IDSs are generally generated on a plaintext data, therefore they can successfully operate in an environment where network traffic is encrypted.
3. Performance of the host-based IDSs is not affected by the topology of the network they operate in. They successfully operate on switched networks.

2.1.1.1.2. Disadvantages of Host-based IDSs

1. As host-based IDSs should be placed on every monitored host, it is harder to manage and configure host-based IDSs.
2. Host-based IDSs run on the host targeted by attacks, and it may be disabled by a successful attack. Such as by certain denial-of-service attacks.
3. As host-based IDSs can only see network packets received by its host, detection performance of host-based IDSs is poor in the case attacks are targeted to the entire network.
4. Amount of information used by host-based IDSs can be huge; hence host–based IDSs may require additional storage on the system running.
5. Host-base IDSs share the computing resources (e.g. CPU, main memory) with the monitored host. Consequently, they cost additional operational overheads and may affect the performance of the hosting computer.

2.1.1.2. Network-based IDSs

As computing environments shifted from mainframe to the networks of workstations, studies on intrusion detection started to focus on attacks targeted to the network. Network attacks can not be detected by examining operating system trails or system logs, or at least detection of network attacks by examining data sources on the host computer is not an easy task. As a result, network-based IDSs were developed, which sniff network packets and search attacks in these network packets. Network-

based IDSs monitor activities on a network segment or switch, so that they can protect hosts connected to the monitored segment. Network-based IDSs generally consist of sensors which are placed at various points (such as at LAN and WAN backbones) in the network. As sensors are limited to run IDS, they can run in "stealth" mode, therefore they can be more secured against attacks.

2.1.1.2.1. Advantages of Network-based IDSs

1. A large network can be monitored easily by employing a few numbers of sensors, if sensors are placed at the critical parts of the network (for example at hubs, routers or probes).
2. Network-based IDSs are generally passive devices and run in stealth mode, hence do not affect the normal operation of the network.
3. Network-based IDSs can be very secure against attacks, and even they can be made invisible to the attackers.

2.1.1.2.2. Disadvantages of Network-based IDSs

1. If monitored network is large or network traffic is high, it may be difficult to process all network packets.
2. Problems arise when network-based IDSs placed on a switched network. Most switches do not provide universal monitoring ports and this fact limits the monitoring ability of network-based IDS.
3. Network-based IDSs can not analyze encrypted traffic. This is due to the fact that, the sensors analyze packet headers to determine source and destination addresses and type of data being transmitted, and analyze the packet payload to discover information in the data being transmitted.
4. Malformed network packets may cause a network-based IDS to crash.

2.1.2. IDS Analysis

Second component of the intrusion detection system is the analysis engine, where decision of whether the monitored activity is an attack or not is taken. Intrusion

detection analysis engines are generally classified into two categories: misuse detection and anomaly detection.

2.1.2.1. Anomaly Detection

Anomaly detectors assume that misuse or intrusions are highly correlated to abnormal behavior of either a user or a system (Ghosh and Schwartzbard 1999). Anomaly detection approaches must first baseline the normal behavior of the object being monitored, and then detect possible intrusions by using deviations from this baseline. These profiles are constructed from historical data collected over a period of time of normal operation. Anomaly detection typically creates knowledge bases containing the profiles of the monitored users, programs or systems. Threshold detection, statistical measures, neural networks and rule-based measures are the techniques used in anomaly detection. A block diagram of a typical anomaly detection system is given in Figure 2.1.

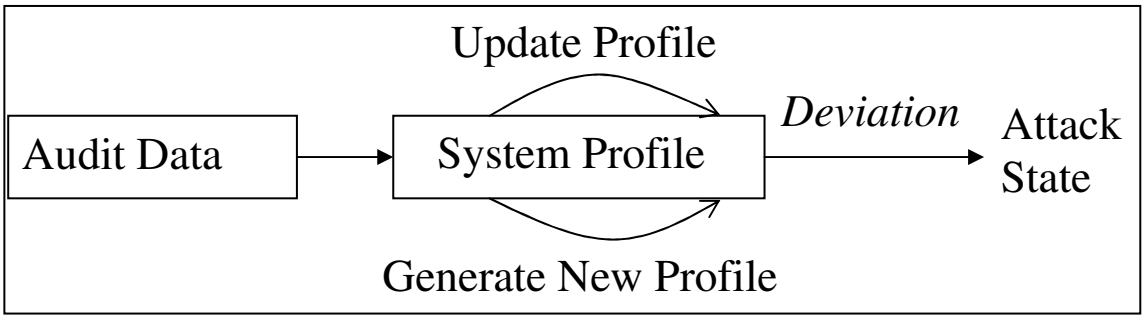

Figure 2.1. Block diagram of a typical anomaly detection system.

2.1.2.1.1. Advantages of Anomaly IDSs

1. As any significant deviation from normal profile will be flagged as anomalous, anomaly detectors can detect unknown attacks.
2. Anomaly detectors do not require constant updating of rules or signatures of novel intrusion.
3. Anomaly detectors can produce information that can in turn be used to define signatures for misuse detectors.

2.1.2.1.2. Disadvantages of Anomaly IDSs

1. The high false positive rate is the main drawback of the anomaly IDSs. This is due to the fact that, the normal profile of a system can not be fully learned and/or behavior of users or programs may change over time.
2. In order to build normal profile of a system, system in question should be monitored and information should be collected, which in turn will be used to draw normal behavior of the system. But if the collected information contains attacks, intrusive behavior will be a part of the normal profile, and in future these attacks will go undetected.
3. Anomaly detection approaches need extensive data sets to build profile of the system.

2.1.2.2. Misuse Detection

Misuse detectors attempt to model attacks on a system as specific patterns, then look for events or sets of events that match a predefined pattern of events that describe a known attack (Ghosh and Schwartzbard 1999). As the patterns corresponding to known attacks are called *signatures*, misuse detection is sometimes called "signature-based detection". Expert systems, signature verification and state transition diagrams are the techniques used in misuse detection. While anomaly detection typically utilizes threshold monitoring, misuse detection techniques frequently utilize a rule-based approach. Today most of the commercial and research intrusion detection tools are misuse detection which are based on attack signatures. A block diagram of typical misuse detection system is given in Figure 1.2.

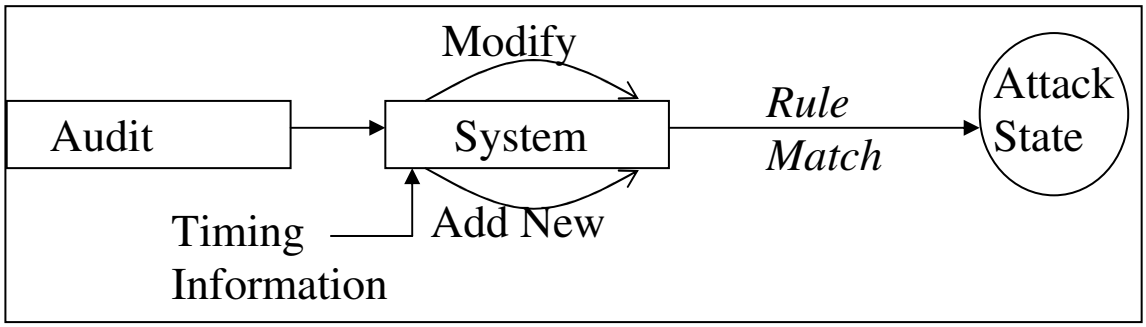

Figure 2.2. Block diagram of a typical misuse detection system.

2.1.2.2.1. Advantages of Misuse IDSs

1. Misuse IDSs can detect intrusion with a certain degree of certainty. Misuse detectors are very effective in detecting attacks without giving high false alarm rates.
2. Misuse IDSs can detect all intrusions whose signatures are known.
3. Misuse IDSs are easy to implement (state machine, signature analysis) and deploy (no need to form a profile of the system).

2.1.2.2.2. Disadvantages of Misuse IDSs

1. Detection ability of misuse detectors is limited to signatures that they posses. A new intrusion or even a variation of a known intrusion may be undetected. So misuse IDs require regular updates of signatures in order to remain current.
2. The process of developing a new attack signature is time consuming.

2.1.3. Response Component

Response component is the third component of an IDS , where reaction to a detected attack is given. According to the response types IDS can be either active or passive. An IDS is said to be active, if it actively reacts to the attack by taking corrective (closing holes) or proactive (logging out possible attackers, closing down services). If an IDS just generates alarms, it is said to be passive. Passive IDS responses provide information to system administrator who takes necessary actions based on that information.

2.2. Neural Networks

Modern digital computers can perform thousands of operations in a second and numerical computation power of human brain is far below the digital computers. However, even the most powerful computers can not compete with humans when performing some perceptual tasks (recognition of a face in crowd for example). Power of human brain, enabling human to outperform the digital computers in completing

some tasks, comes from the fact that human brain is composed of large number of massively interconnected computing units, so called neurons.

Inspired from biological neural networks, artificial neural networks are massively parallel computing systems consisting of an extremely large number of simple processors (called neurons) with many interconnections and coefficients associated with interconnections (weights) (Jain et al. 1996). An artificial neural network (or simply neural network) is a computational model that can be defined by four parameters (Kasabov 1996):

1. Models of neurons.
2. Architecture.
3. Learning algorithm.
4. Recall algorithm.

Neural networks have many characteristics that neither traditional Von Neuman nor modern parallel computers have. These characteristics include massively parallel computation, learning ability, generalization ability, adaptivity, and fault tolerance.

2.2.1. Models of Neurons

Major components of neuronal model are given in Figure 2.3. Three basic elements constitute the neuron model, which are connection links, an adder and an activation function.

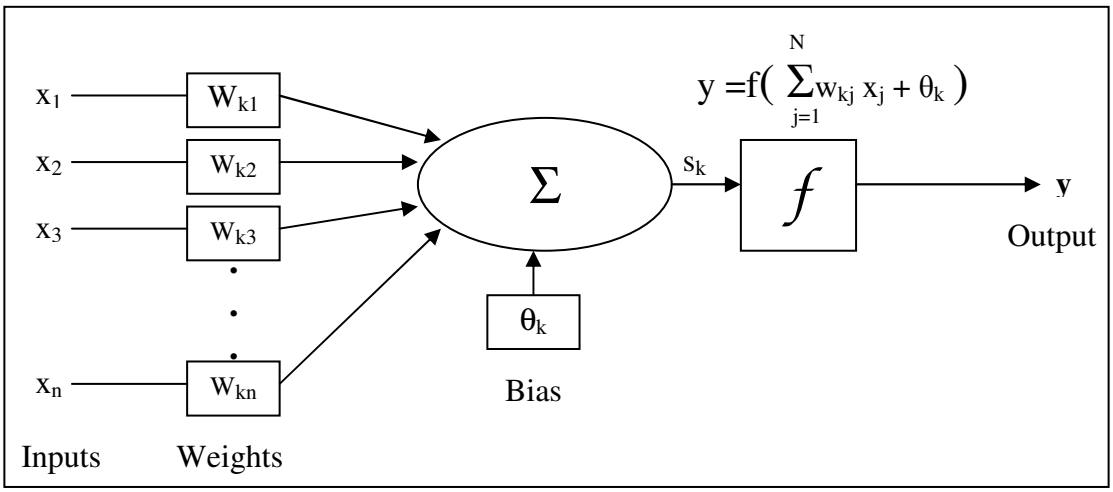

Figure 2.3. Major components of an artificial neuron.

Two neurons are connected with each other with a connection link (or synapse) and each connection link is associated with a weight which determines the computing effect of the neuron on whole neural network. Each of the neurons, x_j, is multiplied with the weight, w_{kj}, associated with the link that connects neuron x_j to another neuron k. These products are summed and presented to an activation or transfer function, which is used to limit the amplitude of the output. The most popular activation functions are given in Figure 2.4.

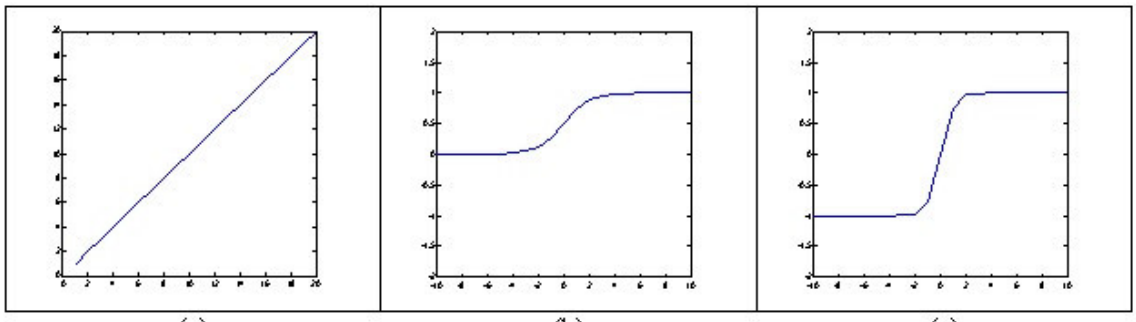

Figure 2.4. Most popular activation functions.

In order to increase or decrease the net input of the activation function, a bias (θ_k) is externally applied to neuron k. In mathematical terms, a neuron k can be described by the following pair of equations:

$$s_k = \sum_{j=1}^{m} w_{kj} \cdot x_j \qquad (2.1)$$

and

$$y_k = f(s_k + \theta_k) \qquad (2.2)$$

where $x_1, x_2, ..., x_m$ are the input signals, w_{k1}, w_{k2}, ..., w_{km} are the synaptic weights of neuron k; s_k is the linear combiner output due to the input signals; θ_k is the bias; f(.) is the activation function; and y_k is the output produced by the neuron.

In order to make formula 2.2 uniform, generally bias is assumed to be an input unit, x_0, which has a weight value, w_{k0}, of 1. Then the formula 2.1 will be

$$s_k = \sum_{j=0}^{m} w_{kj} \cdot x_j \tag{2.3}$$

and

$$y_k = f(s_k) \tag{2.4}$$

2.2.2. Neural Network Architectures

Topology of a neural network describes the organization of connections between neurons. Neurons in a neural network can be fully connected, where every neuron is connected to every other neuron, or can be partially connected.

Architectures of neural networks can be grouped into two categories: (i) Feedforward networks, where data is strictly propagated from input units to output units in feed-forward manner, hence there exists neither feedback connections nor loops in the network. Feedforward neural networks are memoryless systems in the sense that state of the network is not affected by the previous produced outputs; (ii) Feedback (or recurrent) networks contain feed-back connections, hence loops exist within these networks.

In our experiments two different architectures are used. These architectures are multi-layer perceptron network which is a feedforward network and self-organizing maps, which is a recurrent network. In the following sections these architectures are explained in detail.

2.2.2.1. Multi-Layer Perceptron

Multi-layer perceptron (MLP) has an input layer of source nodes and output layer of neurons. These two layers connect the neural network to the outside world; training set is presented to the MLP through input layer and output produced by the neural network exists in the output layer. In addition to these two layers, MLP has one or more hidden layers which can not be directly accessible through the outside world. A sample MLP network with two hidden layers is given in Figure 2.5.

A layer of processing elements makes independent computations on data that it receives and passes the results to another layer. The next layer may in turn make its independent computations and pass on the results to another layer. Finally, a group of one or more processing elements determines the output from the network. Each processing element makes its computation based upon a weighted sum of its inputs.

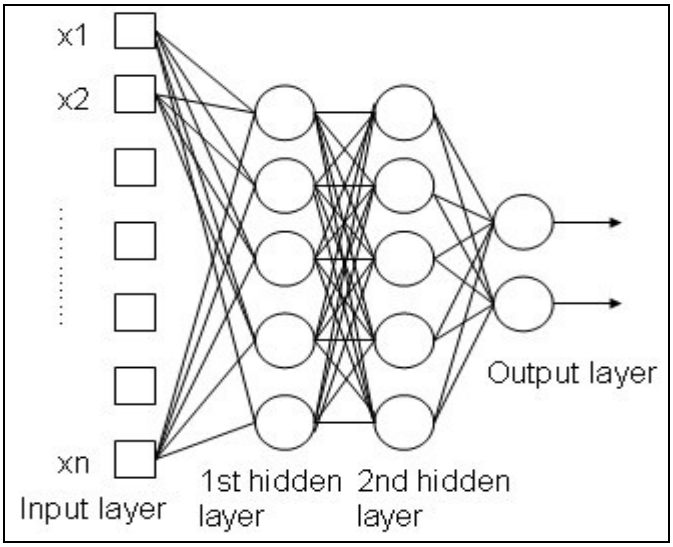

Figure 2.5. A MLP network with two hidden layers.

A L-layer MLP network consists of an input stage, (L-1) hidden layers and an output layer connected in a feedforward manner without any feedback connections between neurons (Jain et al. 1996).

MLP networks can be successfully trained, even if training examples are linearly inseparable (input patterns cannot be separated into the output categories by a single line). One of the most popular training algorithms used in MLPs is backpropagation algorithm, which is described in section 2.2.3.1.

Training process of MLP networks continues until a certain number of iterations or a desired error rate is reached. The most common error approximation method used in MLP networks is mean square error (MSE) and it is defined by the following formula:

$$Err = (d - y)^2 / 2 \qquad (2.5)$$

where d is the desired output for the given input and y is the output produced by the neural network.. A total MSE sums the error over all individual examples and all the output neurons in the network.

$$MSE = \left(\sum_{k=1}^{l} \sum_{j=1}^{m} (d_j^{(k)} - y_j^{(k)})^2 \right) / p.m \qquad (2.6)$$

where $y_j^{(k)}$ is the output value of the j^{th} output of the network when k^{th} training example is presented; $d_j^{(k)}$ is the desired output for the j^{th} output for the k^{th} training example; p is the number of training examples in the training data; and m is the number of output neurons in the neural network. Root-mean square error (RMSE) is the root of the MSE.

2.2.2.2. Self Organizing Maps

The term *self-organizing* refers to that; these networks can adapt their weights without giving any correct answers. Self-organizing maps (SOMs) modify connection weights based on the characteristics of the training set. A SOM consists of two layers, an input layer and an output layer (also called feature map), which represents the output vectors. The weights of the connections of an output neuron *j* to all the other n input neurons form a vector w_j in an n dimensional space. SOMs are trained to cluster input vectors sharing the same features. Similar input vectors are represented by near neurons in the feature map. (Kasabov 1996). The idea behind the learning in SOMs is that, the neuron whose weight vector was closest to the input vector is updated to be even closer. An example of SOM network is given in Figure 2.6.

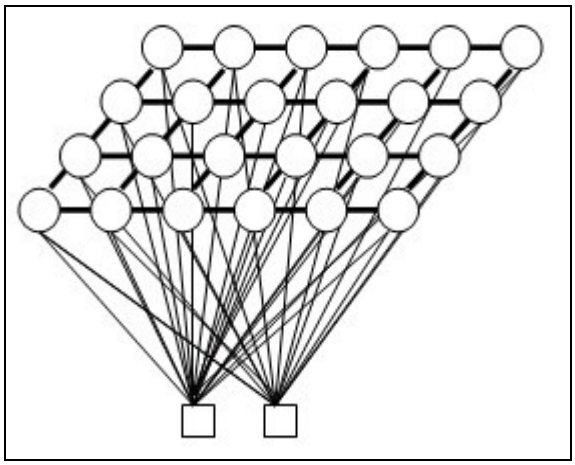

Figure 2.6. An example of self-organizing map network.

In the first step of the learning, all weight vectors are initialized such that values of all weight vectors are assigned differently. From the set of input vectors, a sample input vector, *x*, is selected and each neuron computes Euclidean distance (which is the straight line distance between two points) between its weight vector and sample input vector (Formula 2.7).

$$i(x) = \arg\min_j \left\| x(n) - w_j \right\|, \quad j = 1,2, ...,l \tag{2.7}$$

The weight vector which has the minimum Euclidean distance, hence most similar to sample input vector, is called *best-matching* or *winning neuron*.

Only the winning neuron and its neighbors gain the right to update their weights. Weight vectors are updated by the following formula:

$$w_j(n+1) = w_j(n) + \eta(n) h_{j,i(x)}(n)(x(n) - w_j(n)) \tag{2.8}$$

where $\eta(n)$ is the learning-rate parameter, and $h_{j,\ i(x)}(n)$ is the neighborhood function around the winning neuron $i(x)$. Both $\eta(n)$ and $h_{j,\ i(x)}(n)$ changes dynamically during the learning process. While initial neighborhood size is often set to 1/2 to 2/3 of the network size, it shrinks exponentially with time.

2.2.3. Learning in Neural Networks

In the context of neural networks, learning is the process of updating connection weights so that the neural network can perform a desired task efficiently. A neural network is trained so that it produces set of desired output vectors Y, from a given set of input vectors X. The set X, which is used to train the neural network, is called training set, and elements of X are called training examples. Performance is improved over time by iteratively updating the weights in the network.

There are three main learning algorithms (Jain et al. 1996):

1. *Supervised*. In supervised learning, beside from the training set, desired outputs of the given training set is presented to the neural network. Aim of the supervised learning is minimizing the difference between the output

produced by the network and the desired output by altering the values of the connection weights.

2. *Unsupervised*. Correct answers associated with each training example is not required in unsupervised learning and only training set is presented to the neural network. In unsupervised learning, correlations within the training set are explored, and data are clustered based on these correlations. Self-organizing maps utilize unsupervised learning.
3. *Hybrid*. Hybrid learning combines supervised and unsupervised learning. Some parts of the weights are updated in supervised manner and the others are updated through unsupervised learning.

2.2.3.1. Backpropagation Algorithm

One of the most popular supervised learning algorithms is the back propagation algorithm. The algorithm consists of two phases: *Forward phase and backward phase*. In the forward phase, first, the weights of the network are randomly initialized. Then, the input signals are propagated through the network. Afterwards, the output of the network is calculated and compared to the desired value. In the end of the forward phase, the error of the network is calculated. Error of the output neuron i (e_i) is calculated by the formula:

$$e_i = d_i - y_i \tag{2.9}$$

where d_i is the desired response and y_i is the output produces by the neural network in response to the input x_i.

Aim of the backpropagation algorithm is to reach global minimum value (Figure 2.7) on the error surface.

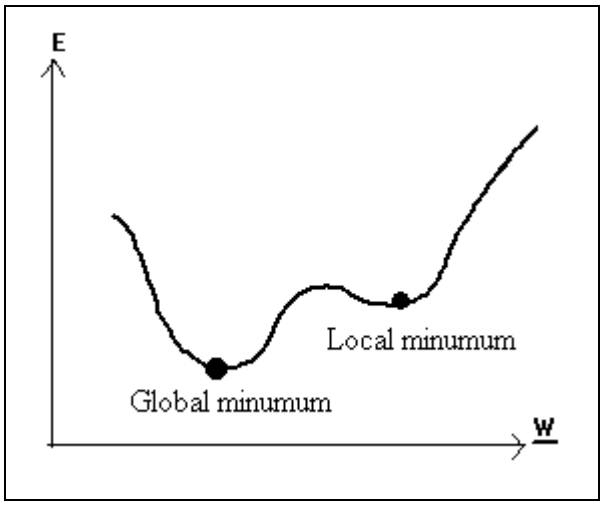

Figure 2.7. Global and local minimum in error surface.

In backward phase, calculated error signal is propagated backward and in order to minimize the error, weights are updated. Change in weights can be calculated by gradient descent learning rule (Haykin 1999).According to the gradient descent learning rule, correction applied to the weight w_{ji} at the iteration *n* is denoted by $\Delta w_{ji}(n)$, and calculated by

$$\Delta w_{ji}(n) = \eta \delta_j(n)\, y_i(n) \tag{2.10}$$

where η is a numerical constant (learning-rate parameter of the backpropagation algorithm) and $\delta_j(n)$ is local gradient.

Local gradient of output neurons is equal to the product of the derivative of activation function, $f'(\cdot)$, and error signal, $e_j(n)$, and defined by

$$\delta_j(n) = e_j(n) f'(s_j(n)) \tag{2.11}$$

Local gradient of the hidden neurons is associated with derivative of the activation function and local gradient of the next hidden layer or output layer. Local gradient for neurons in hidden layers is defined by

$$\delta_j(n) = f'(s_j(n)) \sum_k \delta_k(n)\, w_{kj}(n) \tag{2.12}$$

Learning rate parameter, η, is used to abridge the training time. But if the learning rate parameter is chosen too high (e.g. 0.9), algorithm oscillate between local minimums, and may not achieved to reach the global minimum, whereas selecting learning rate too small results in long training periods. One way to speed up the learning when learning rate is chosen small or avoid oscillation between local minimums when learning rate is chosen to big is to utilize a parameter, *momentum*. By introducing the momentum parameter, change in weight, $\Delta w_{ji}(n)$, is made dependent to the previous weight change, $\Delta w_{ji}(n-1)$. Modified backpropagation algorithm which uses momentum, α, is given in formula 2.13.

$$\Delta w_{ij}(n) = \eta \delta_j(n) y_i(n) + \alpha \Delta w_{ij}(n-1) \tag{2.13}$$

After the training was completed, connection weights are frozen. Afterwards, in order to validate whether the neural network was trained sufficiently or not, a test set, which is not part of the training set, was presented to the trained network and its performance is evaluated.

Backpropagation algorithm is simple to implement. However when dealing with difficult learning tasks, training time of the backpropagation networks can be lengthy and even algorithm may not converge to the desired error rate.

2.2.3.2. Levenberg-Marquardt Algorithm

Levenberg-Marquardt is an advanced non-linear optimization algorithm (Hagan and Menhaj 1994). It can be used to update the weights in the network just as backpropagation algorithm. It is reputably the fastest algorithm available for such training.

The Levenberg-Marquardt algorithm is designed specifically to minimize the sum-of-squares error function, using a formula that (partly) assumes that the underlying function modeled by the network is linear. A move is only accepted if it improves the error, and if necessary the gradient-descent model is used with a sufficiently small step to guarantee downhill movement. The weight update vector $\Delta \underline{x}$ is calculated as

$$\Delta \underline{x} = \left[J^T(\underline{x}) J(\underline{x}) + \mu I \right]^{-1} J^T(\underline{x}) \varepsilon \tag{2.14}$$

where ε is the vector of errors, μ is the learning rate parameter, and J(x) is the Jacobian matrix that is the matrix of partial derivatives of the errors with respect to the weights. Jacobian matrix can be calculated with the following formula:

$$J(\underline{x}) = \begin{bmatrix} \frac{\partial e_1(\underline{x})}{\partial x_1} & \frac{\partial e_1(\underline{x})}{\partial x_2} & \cdots & \frac{\partial e_1(\underline{x})}{\partial x_n} \\ \frac{\partial e_2(\underline{x})}{\partial x_1} & \frac{\partial e_2(\underline{x})}{\partial x_2} & \cdots & \frac{\partial e_2(\underline{x})}{\partial x_n} \\ \cdots & & & \\ \frac{\partial e_m(\underline{x})}{\partial x_1} & \frac{\partial e_m(\underline{x})}{\partial x_2} & \cdots & \frac{\partial e_m(\underline{x})}{\partial x_n} \end{bmatrix} \quad (2.15)$$

Levenberg-Marquardt outperforms the basic backpropagation and its variations with variable learning rate in terms of training time and accuracy. However the computation and memory requirements of the algorithm are high.

2.3. DARPA 1999 Intrusion Detection Evaluation

In 1998 and in 1999, the U.S. Defense Advanced Research Projects Agency (DARPA) made an evaluation among commercial and research intrusion detection research projects. To date, it is the most comprehensive scientific study known for evaluating different intrusion detection systems. DARPA study highlights the strengths and weaknesses of current research approaches to intrusion detection. The DARPA scientific study is the first of its kind to provide independent third party evaluation of intrusion detection tools against such a large corpus of data.

The 1999 DARPA Off-Line Intrusion Detection Evaluation had four main objectives which were as follows (Haines et al. 2001):

- Support developers of intrusion detection systems, by providing rich data sets for testing and experimentation. This data greatly facilitate algorithm development by providing examples of normal and attack traffic and eliminating the additional effort of traffic generation.
- Evaluate intrusion detection approaches by analyzing the strengths and weakness of each.

- Analyze differences between alternate approaches and determine the reasons of false positives and false negatives.
- Report summarized results of the evaluation and conclusions from the detailed analysis of each intrusion detection system in evaluation to DARPA to guide future efforts.

MIT's Lincoln Laboratory set up a simulation environment for generating and distributing sniffed network data and audit data recorded on host machines. Software automata simulated hundreds of programmers, secretaries, managers, and other types of users running common UNIX application programs and some Windows NT programs. In addition, a small number of actual hosts appeared as if they were thousands of hosts with different IP addresses. Thus, hundreds of users on thousands of hosts were represented in the simulation. DARPA 1999 Intrusion Detection Evaluation testbed is given in Figure2.8.

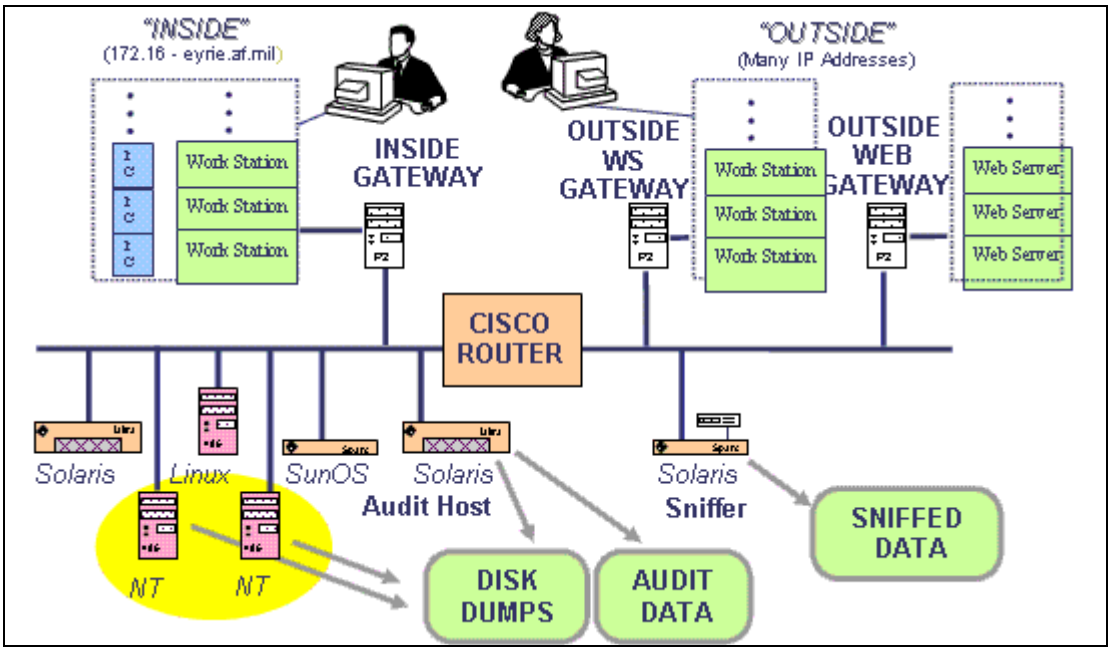

Figure 2.8. DARPA 1999 Intrusion Detection Evaluation Testbed.
(Source: Haines et al. 2001)

Because all the data were generated, the laboratory has a priori knowledge of which datum represents normal and which represents attack. Network traffic was generated to represent the following types of services: HTTP, SMTP, POP3, FTP, IRC, Telnet, X, SQL/telnet, DNS, finger, SNMP, and time. More than 200 instances of 58 attack types were launched against victim UNIX and Windows NT hosts in three weeks

of training data and two weeks of test data. This corpus of data is the most comprehensive set known to be generated for the purpose of evaluating intrusion detection systems.

CHAPTER 3

INTRUSION DETECTION WITH NEURAL NETWORK

This chapter provides the reason for building neural network based intrusion detection system. In addition, similar researches related to study, advantages and disadvantages of neural network based intrusion detection system are presented in this chapter.

3.1. Motivation

Although misuse and anomaly IDSs improve the security of an information system to a certain extent, both of them have limitations.

Most current approaches to the process of misuse detection utilize some form of rule-based analysis. Rule-Based analysis relies on sets of predefined rules that are provided by an administrator, automatically created by the system, or both. These rules are used by the system to make conclusions about the security-related data from the intrusion detection system. Unfortunately, the detection ability of misuse systems is limited to the rule base that they posses. Hence misuse detectors require frequent updates to remain current. The required updates may be ignored or performed infrequently by the administrator and this may lead the system vulnerable to the attacks. In addition, writing a rule or signature of a new attack is not an easy task and can be time consuming.

Another limitation of misuse detectors is that the misuse intrusion detection systems do not have generalization property and hence fail to detect unknown and even variations of known attacks, thus misuse IDSs generally have high false negative rates.

Anomaly detectors also have limitations. For instance, although anomaly detectors can detect an attack accurately, they can not identify the specific type of attack occurring. However, the most significant problem of anomaly detection approach is the high false positive rates. Any deviation from the baseline will be flagged as intrusion; legitimate behavior outside the baseline will be labeled as intrusive. Another problem

arises if an attack occurs during the establishment of the baseline, then this intrusive behavior will be the part of the normal baseline.

Instead of defining rules for each of the attack, which is a troublesome and time consuming process, neural networks can distinguish malicious and normal activity by learning correlations between training patterns. By applying neural networks to intrusion detection, the need for managing rules and signatures can come to end. In addition to that, neural networks posses the ability of generalization so that they can generalize from the previously observed behavior to similar future behavior. A neural network-based intrusion detection system could potentially address novelty detection problem in misuse detection systems and could lower the false positive rates of anomaly detectors to an acceptable level.

Another reason to employ neural networks in intrusion detection is that, neural networks can cluster patterns which share similar features, thus the classification problem in anomaly detectors can be solved by neural networks.

3.2. Earlier Work

- One of the first attempts to employ neural networks to intrusion detection was performed by Ryan et al. in 1998 (Ryan et al.1998). An offline intrusion detection system, Neural Network Intrusion Detector (NNID), was trained and tested on a system with 10 users.

The NNID anomaly intrusion detection system is based on identifying anomaly and legitimate usage based on the distribution of commands that user executed. For each user in the system, user profile was formed based on the commands executed by the user. System was implemented on UNIX environment and operating system logs were used to extract command usage. A set of 100 most common used commands in the logs and their frequencies were used to form input vectors of the neural network.

The standard two-layer backpropagation architecture was chosen for the neural network. The input layer consisted of 100 units, representing the user vector; the hidden layer had 30 units and the output layer 10 units, one for each user. The network implemented in the PlaNet Neural Network simulator. The system was able to detect 96% of attacks. The achieved false positive rate was 7%.

- As most computer violations are made possible by misusing of computer programs, host-based intrusion detection approaches are shifting from inspecting user behavior to inspecting program behavior. Rather than building user profiles, Ghosh et al. (Ghosh and Schwartzbard 1999) presented a host based intrusion detection system that focused on building program profiles and used these program profiles to recognize normal software behavior and malicious software behavior. In order to monitor process behavior, operating system was monitored to capture the system calls. The system was trained and tested on SUN platform and use Basic Security module (BSM) as source of data. Input data were extracted from BSM and a distance metric, which constituted input vectors of the neural network, was calculated for each data item and exemplar strings.

The IDS presented was a single hidden layer MLP. The number of input nodes was equal to the number of exemplar strings. There was a single continuous node in the output layer. For each program, networks were trained with 10, 15, 20, 25, 30, 35, 40, 50, and 60 hidden nodes; the network, which gave the minimum false negative and false positive rates, was kept and others were discarded. Lucky Bucket algorithm is used to capture the temporal locality of anomalous events.

Performance analysis was done with DARPA database. With a leak rate of 0.7 the anomaly detection system achieved 77.3% detection rate and false positive rate of 2.2%, whereas with the same leak rate misuse detection system achieved 90.9% detection rate and false positive rate of 18.7%.

- Lee and Heinbuch (Lee and Heinbuch 2001) utilized an experimental intrusion detection system with a hierarchy of neural networks. Each of the neural networks in the hierarchy focused on different portions of nominal TCP behavior. Portions of these observed TCP behaviors are connection establishment, connection termination and port usage. System was trained to detect three kinds of attack, which are SYN flood, fast SYN port scan, and stealth SYN port scan.

Backpropagation learning algorithm was used to train system. Input vectors to each of the neural networks were generated randomly. Number of input vectors generated was 4000-6000, which is claimed to be sufficient to train the system.

- Lippmann and Cunnigham of MIT Lincoln Laboratory (Lippmann and Cunningham 2000) conducted a misuse detection model with neural networks, by searching attack specific keywords in the network traffic (Figure 3.1.). They used a MLP network to detect Unix-host attacks, and attacks to obtain root-privilege on a

server. The data that they presented to the neural network consisted of attack-specific keyword counts in network traffic. Two neural networks were used in the system, one for providing an attack probability and one for classifying attacks.

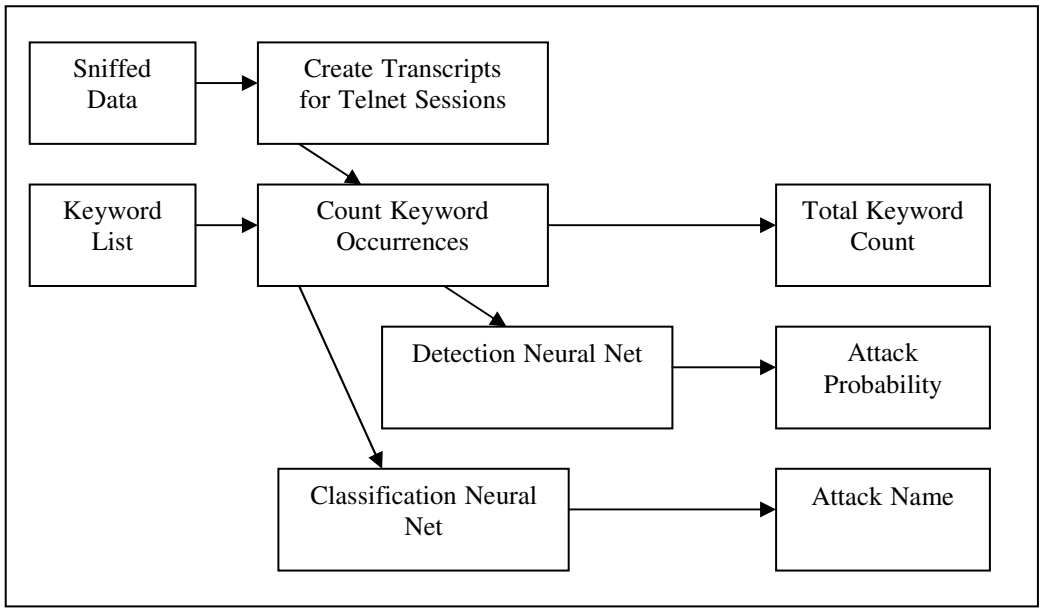

Figure 3.1. Block diagram of enhanced intrusion detection system.

A two-layer perceptron was designed with k input nodes, 2k hidden nodes and 2 outputs (normal and attack) and the training algorithm used in the system was backpropagation.

Sniffed network packets were first processed to construct transcripts containing all bytes transmitted to and from victim hosts during telnet sessions. These transcripts were then processed to get counts of each keyword. This count, which is the first output of the system, would be used as reference to provide a probability of an attack in the telnet session. In addition to providing an attack probability, this count is also used to classify known attacks, thus provide an attack name.

Using neural network to weight keyword counts of a smaller set of 30 keywords lowers the false alarm rate to an acceptable and practical rate of roughly one false alarm per day with detection rate of %80.

- In 2002 Jirapummin, Wattanapongsakorn and Kanthamanon (Jirapumin et al. 2002) presented an alternative methodology for both visualizing intrusions by using self-organizing map and classifying intrusions by using Resilient Propagation Neural

Network (RPROP). Their research was focused on detection of TCP SYN flooding and port scanning attacks.

Normal data set, Neptune attack (SYN flooding), Portsweep and Satan attacks (port scanning) were selected from KDD Cup 1999 data set. Total of 121.820 training patterns were divided equally into 8 sets. Each set is then clustered by a 1.234-unit SOM network. For RPROP, 3 layer network is utilized where there were 70 nodes in first hidden layer, 12 neurons in second hidden layer and 4 neurons in the output layer. The transfer functions for the first hidden layer, second hidden layer and the output layer of RPROP were tan-sigmoidal, log-sigmoidal and log-sigmoidal respectively.

Two different datasets were used for testing purposes. First test set contained 98.648 data, which was captured from the same network as the training data. Second test set contained 126.372 unseen normal and attack data from a different network.

From the IDS simulation results, 90% detection rate with less than 5% false alarm rate was achieved in three selected attack programs.

- Bivens et al. (Bivens et al. 2002) proposed a neural network model for a network-based intrusion detection system. Proposed IDS was an anomaly detection system and MLP network was used for detection. System read tcpdump data and sent to a preprocessing unit to keep the statistics of the network traffic. System used time-window method, such that traffic intensities were analyzed at fixed time intervals. It was necessary to group similar traffic trends, and therefore preprocessed network traffic was sent to a clustering module, in which self-organizing maps were used for clustering purposes. Clustered traffic then sent to a normalization module for formatting, and output of the normalization module was driven to neural network.

DARPA 1999 training dataset was used for testing the system. System was tested against denial of service attacks, distributed denial of service attacks, and port scans. Union of all attacks, system performed detection with 76% false positive and with no false negatives.

- In another study accomplished by Ghosh et al. (Ghost et al. 1999), program behaviors were created from system calls and intrusion detection was performed from these profiles. An Elman network, which is similar to MLP with additional context nodes, was employed to perform classification of short sequences of BSM events. A leaky bucket algorithm, which provides some memory of recent events, was used to reduce false positive rate. Randomly generated data were used to train the neural network to identify normal an anomalous data. Performance of the system was tested

with data provided by the DARPA Intrusion Detection Evaluation. With a leak rate of 0.7, the Elman Network was able to detect 77.3% of all attacks without any false alarms.

- In another study (Zhang et al. 2001), statistical analysis was used in conjunction with MLP networks. System is a distributed hierarchical application in the sense that system consists of hierarchy of Intrusion Detection Agents (IDAs) at multiple tiers where each tier corresponds to different network scope. IDAs are IDS components that monitor the activities of a host or a network.

A diagram of an IDA is illustrated in Fig. 3.2., which consists of the following components: the probe, the event preprocessor, the statistical processor, the neural network classifier and the post processor.

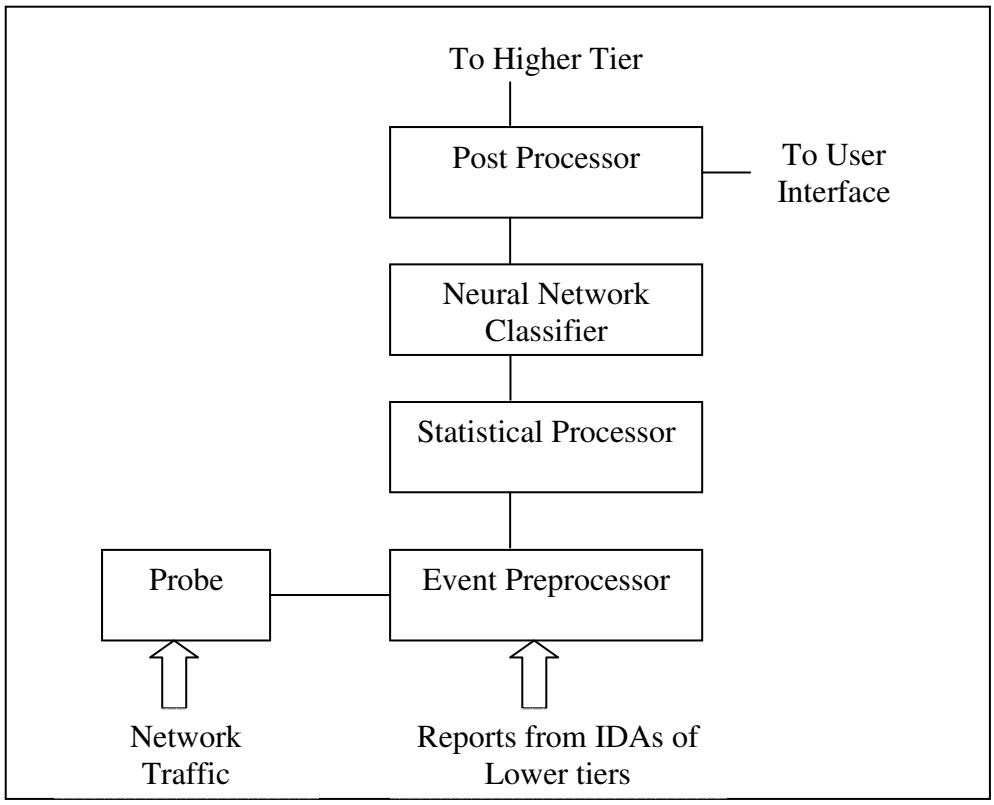

Figure 3.2. A diagram of an IDA.

Probe collects network traffic and abstracts it into statistical variables. Event preprocessor collects data from probes and other agents and formats it for the statistical analyzer. Statistical model compares the data to the previously compiled reference model which describes the normal state of the system. A “stimulus vector” is formed

and forwarded to the NN. Neural network analyzes the vector and decides whether it is anomalous or normal. Post processor generates reports for the agents at higher tiers or it may display the results through a user interface.

Backpropagation , perceptron, perceptron-backpropagation hybrid, fuzzy ART MAP, radial-basis function networks with 2-8 hidden nodes were tested. The experimental testbed consisting of 11 workstations and 1 server was built by using OPNET network simulation software. UDP flooding attack was simulated within the testbed. 10.000 records of network traffic were collected, 6000 of which were used for training and the other 4000 records were used for testing. The system was trained for 100 epochs. The results showed that backpropagation and perceptron-backpropagation hybrid networks outperformed perceptron, fuzzy ARTMAP and radial-basis function networks.

- In another study (Rhodes et al. 2000), it is proposed to use of self-organizing neural networks to recognize anomalies in network data stream. Unlike from other approaches which use self organizing maps to process entire state of a network or computer system to detect anomalies, proposed system breaks down the system by using collection of more specialized maps. A monitor stack (Figure 3.3.) was constructed and each neural network become kind of specialist to recognize normal behavior of a protocol and raise an alarm when a deviation from normal profile occurs. While each protocol attentive to different parts of network packets, different vectorization techniques were employed for each map.

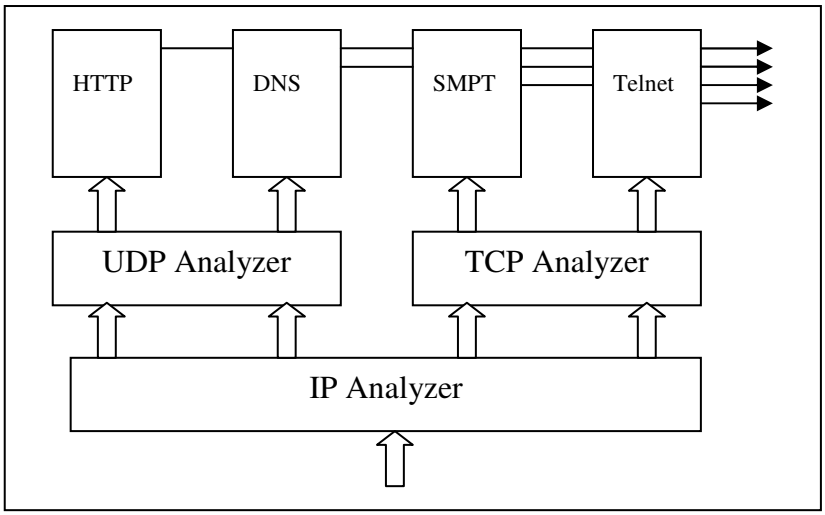

Figure 3.3. General schematic for the Network Monitor Stack.

In the experiment DNS protocol was inspected because of its relatively simplicity. Both of the test intrusions were buffer overflow attempt. Of the forty packets, first thirty were used to train map and the rest of them was used in testing. Of the seven packets transmitted to accomplish the bind4-9-5 exploit, four of the packets successfully detected. In addition, system successfully identified rotshb exploit.

- Cannady (Cannady 1998) developed a network based neural network intrusion detection system where detection was achieved on the packet level. Nine of the packet characteristics of network data were selected and presented to the MLP network which has four fully connected layers. After 3 levels of preprocessing data was normalized, data fields were grouped and converted to a neural network readable format. In addition every packet was labeled whether the packet is an attack or not.

Utilized neural network has nine input nodes and two output nodes and number of nodes in hidden layer was determined empirically. 10,000 packets, of which approximately 1000 were simulated attacks, were collected by RealSecure network monitor. Backpropagation algorithm was used to train neural network. 1000 randomly selected packets were used to test neural network and the remaining were used to train neural network. At the conclusion, root mean square error of 0.0582 was achieved for training data, while root mean square error of 0.069 was achieved for test data.

A brief summary of earlier studies is given in 3.1.

Table 3.1. Summary of Earlier Studies

	Year	**Data Source**	**NN Structure**	**Test Results**
Ryan et al.	1998	Operating System Logs	2-Layer MLP	7% False Positive 4% False Negative
Cannady	1998	Network Packets Collected by Real Secure Network Monitor Software	2-Layer MLP	RMSE of 0.0582 for Training Data RMSE of 0.069 for Test Data.

(cont. on next page)

Table 3.1 (cont.)

Ghosh and Schwarztbard	1999	Sun's BSM	2-Layer MLP	Anomaly Detection: 2.2% False Positive 22.7% False Negative Misuse Detection: 18.7% False Positive 9.1% False Negative
Ghosh et al.	1999	Sun's BSM	Elman Networks	No False Positives 22.7% False Negative
Rhodes et al.	2000	Network Packets	SOM	Prototype could identify packets transmitted to accomplish bind4-9-5 and rotshb exploit
Lippmann and Cunnigham	2000	Network Packets	2-Layer MLP	One False Alarm per Day 20% false Negative
Zhang et al.	2001	Network Packets Generated by OPNET Network Simulation Software	Backpropagation Perceptron Perceptron-Backpropagaiton Hybrid Fuzzy ART MAP Radial Basis Function	Radial Basis Function and Backpropagation networks with 8 nodes in hidden layer outperformed other neural networks. Backpropagation network having 8 nodes in hidden layer had RMSE value less than 0.05 and misclassification rate less than 00.2. Radial Basis Function network having 8 nodes in hidden layer had RMSE value less than 0.05 and misclassification rate less than 0.1.
Lee and Heinbuch	2001	TCP packets	Hierarchy of Neural Networks	-

(cont. on next page)

Table 3.1 (cont.)

Jirapummin et al.	2002	KDD Cup 1999 Dataset	3-Layer RPROP	5% False Positive 10% False Negative
Bivens et al.	2002	DARPA 1999 Dataset	SOM for Clustering MLP for Detection	76% False Positive No False Negatives

3.3. Advantages of NN Based IDS

The most important advantage of neural network based intrusion detection system is that, system will have generalization ability. From the training examples presented during training, neural networks can learn malicious activity flowing through the network and can generalize the observed novel activity from past activities used in training.

Instead of following a set of rules specified by human experts, neural networks can learn input-output relationships from a given set of training patterns. This is the one major advantage of neural networks over rule-based systems. Thus, unlike misuse IDSs, neural network based IDSs do not need regular signature updates.

Without any prior knowledge, neural networks can cluster data which share same features. By examining relationships between data patterns, neural networks can classify patterns successfully.

Another advantage in the utilization of a neural network in the detection of misuse would be the flexibility that the neural network would provide. A neural network is capable of analyzing the data from the network, even if the data is incomplete or distorted.

3.4. Disadvantages of NN Based IDS

One of the most major problems with neural network is determining the number of hidden neurons and more generally determining the structure of the neural network as there is no mathematical method for selection of the neural network structure. Thus, various neural networks with different structures should be trained and the structure and number of the hidden nodes should be determined empirically by selecting the neural networks giving the best performance.

As the training set is the one of the major elements affecting the performance of the neural network, selecting the training set is a crucial issue. Neural network would not be trained, if the training examples are not appropriately selected or the number of training examples is insufficient. In addition, a complex structured neural network or huge number of training examples can result in long training periods.

Although neural networks can be trained sufficiently, it is not always guaranteed that produced outputs will be perfect or even correct. Output produced by a neural network is just an approximation of the desired solution and there will be always certain error.

CHAPTER 4

METHODOLOGY

This chapter presents in detail the research methodology used to build proposed network-based neural network intrusion detection system. In order to build a network-based neural network intrusion detection system, certain steps should be taken (Figure 4.1). First step to be taken is collecting data used to train and test neural networks. We used 1999 DARPA Intrusion Detection Evaluation dataset to train and test neural networks. But number of attack instances is insufficient for training, so attack packets were generated in simulation environment. Second step is preprocessing the collected data, which is in binary tcpdump format, to a neural network readable format. Third step is determining the neural network structure, which is actually determining the number of hidden layers, number of hidden nodes in each layer, activation functions used in neural networks and training algorithm. Fourth step is training neural networks until a certain number of iterations or a certain RMSE value reached. Fifth and the final step is testing the neural network. Trained neural networks tested against training dataset in order to examine how well trained networks learned the data in the dataset. In addition to the training dataset, trained neural networks were tested against dataset which is not part of the training process. Aim of this test is to evaluate the generalization ability of the neural networks.

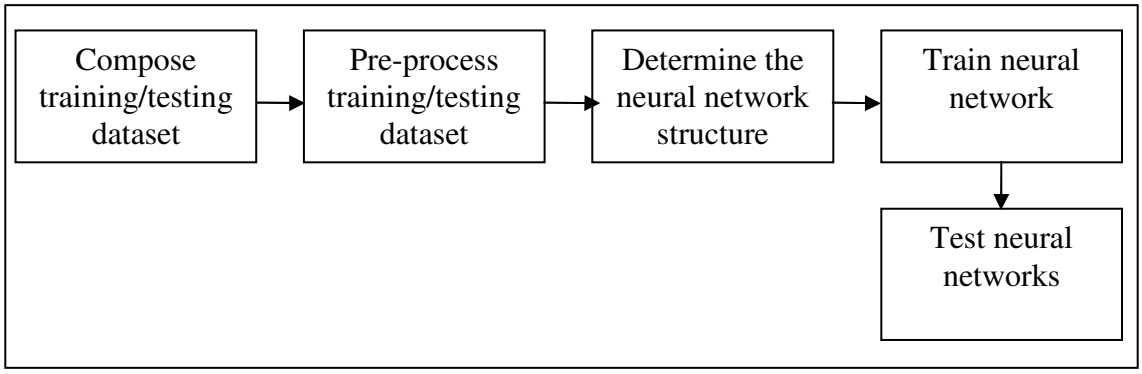

Figure 4.1. Steps to be taken in order to build neural network based IDS.

Neural network simulation was written with MATLAB 6.5. The reason for selecting MATLAB programming language is the ease of generating, training and testing the neural networks with its Neural Network Toolbox.

4.1. Compose Training/Testing Data

As effectiveness of the neural network mainly depends on the training data, collecting data for training and testing is a crucial issue. Data can be obtained by one of the following three ways: by using real traffic, by using sanitized traffic and by using simulated traffic (Mell et al. 2003).

The first option to obtain training/testing data is collecting actually real data and attacking an organization. Although packets would be real, it was unacceptable to attack an organization. In addition to that, privacy of the users in the organization would be violated as private e-mails, passwords and user identities would be released.

In order to overcome security and privacy problems of using real traffic, sanitized traffic was proposed to be used by removing any sensitive data from the data stream. Then attack data can be inserted into the sanitized traffic. The advantage of this approach is that the data can be freely distributed. Nevertheless, the below explained problems arise when using this approach. First of all, most of the content of the background activity may be removed by the sanitization attempt. Next, it is still possible to release sensitive data since it is infeasible to verify large corpus of data.

The third and the most common way to obtain data is to create a testbed network and generate background traffic on this network. In the testbed environment, background traffic is generated either by using complex traffic generators modeling actual network statistics or by using simpler commercial traffic generators creating small number of packets at a high rate. Advantage of this approach is that data can be freely distributed as it does not contain any sensitive information. Another advantage of this approach is that is guaranteed that generated traffic does not contain any unknown attacks as the background traffic is created by simulators. However difficulties exist when using this approach too. Firstly, it's very costly and difficult to create a simulation. Next, in order to model various networks, different types of traffic is needed.

In order to avoid dealing with difficulties of all three approaches, DARPA 1999 Intrusion Detection Evaluation dataset was used for training/testing data. From the attack free first week data, days of Monday, Tuesday and Wednesday were used for training and testing purposes. Dataset of Monday contains 1.272.249 TCP and 127.249 UDP packets. Dataset of Tuesday contains 849.119 TCP and 50.338 UDP packets and dataset of Wednesday contains 649.852 TCP and 45.859 UDP packets. However attack instances in DARPA 1999 Intrusion Detection Evaluation dataset are not sufficient to train the neural network, thus a simulation environment was formed and additional malicious traffic was aimed to obtain as explained below.

Malicious traffic was created in the laboratory. Attacks, which form malicious traffic, were selected based on SANS top 10 vulnerabilities list. Simulation environment in laboratory contains one Linux Server, one Windows Server, two Windows workstations and two Linux workstations. By using network security scanners (e.g. Nessus, Shadow Security Scanner) and attack tools, one Linux and one Windows workstation were used to generate malicious network traffic in addition to the normal traffic. Simulation environment used to generate malicious traffic is given in Figure 4.2.

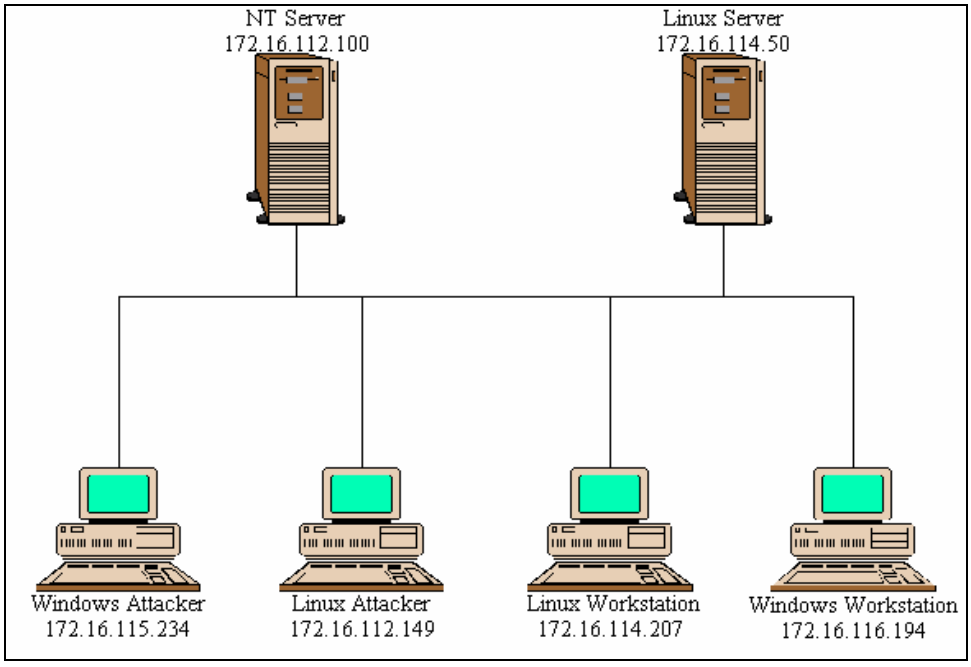

Figure 4.2. Simulation environment used to generate malicious traffic.

Two distinct datasets were needed, one for training the neural network, and one for testing the neural network. For the training dataset, 9450 network packets were

selected from days of Monday and Wednesday from the attack free first week data of DARPA 1999 Intrusion Detection Evaluation dataset. In addition to the attack free traffic, 971 attack packets were generated within the simulation environment and included to the training dataset. For the test dataset, 4233 network packets were selected from day of Tuesday from the first week data of DARPA dataset. 73 malicious network packets were generated and added to the test dataset. Summary of training and testing datasets is given in Figure 4.3.

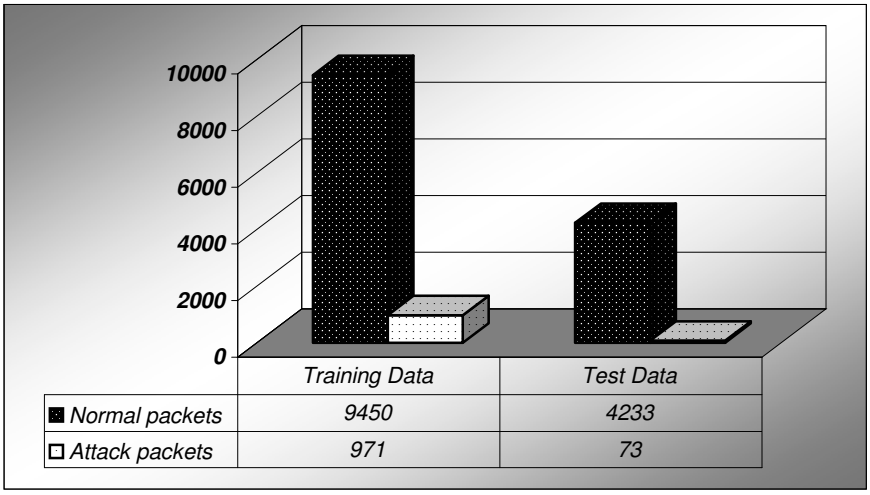

Figure 4.3. Summary of training and testing datasets.

4.2. Preprocess Training/Testing Data

All the data collected from simulation environment and DARPA dataset was in the binary tcpdump format. In order to preprocess and then present them to the neural network, obtained binary tcpdump files should be converted to a neural network readable format. To achieve this goal, an open source intrusion detection tool, Snort, was used.

Snort (WEB_3 2004) is a lightweight network intrusion detection system, capable of performing real-time traffic analysis and packet logging on IP networks. Its author, Marty Roesch, originally designed Snort to be a personal tool which can be used in network traffic analysis and it was originally designed to decode binary tcpdump data into a human-readable form. Snort can be used as a sniffer, packet logger, or network intrusion detection system. In sniffer mode, Snort captures network packets flowing in a shared network and prints the captured network packets to the screen. In packet logger

mode, Snort captures packets in a similar sniffing fashion, but logs the captured data instead of printing it to the screen. In intrusion detection mode, Snort captures network packets like sniffer and logger mod, but applies pre-defined rules on all captured packets. If a packet matches a rule, then it is logged or an alert is generated. One of the most useful aspects of the Snort is that, it supports various relational databases like MySQL, Oracle or SQL Server. Captured packets can be stored in the supported databases. ER diagram of the database created by Snort is given in Figure 4.4.

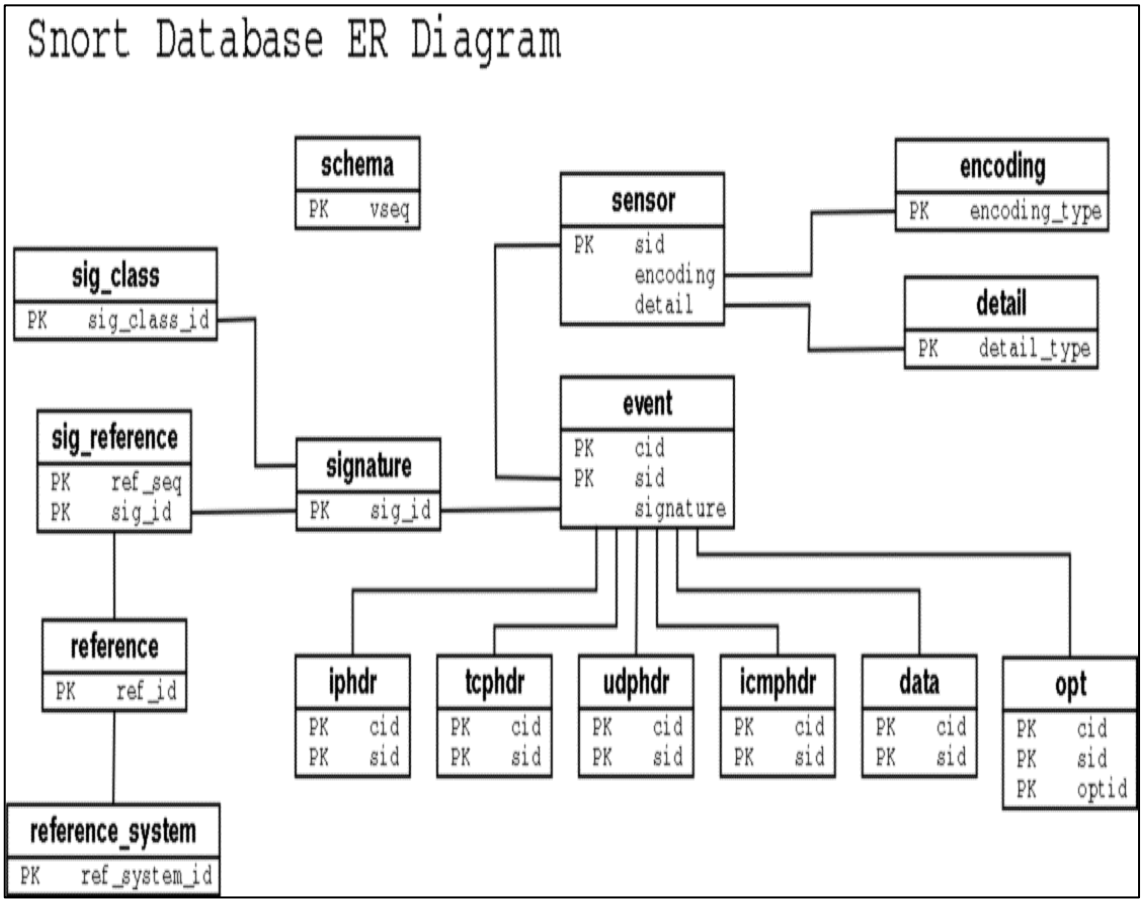

Figure 4.4. ER diagram of the database created by Snort.
(Source: http://www.snort.org/docs/snortdb.png)

Snort version 2.2 with SQL Server support was utilized and configuration file of the Snort was configured so that not only all the packets in binary tcpdump files are logged to the database but also packets containing attacks were labeled. After all network packets in tcpdump files were transferred to the database, stored data must be preprocessed before they are presented to the neural network. Three levels of preprocessing will be conducted:

In the first level of the preprocessing, for each collected packet stored in the database, seven characteristics of a packet will be retrieved. The seven elements were selected because they are typically present in network data packets and they provide a complete description of the information transmitted by the packet (Cannady 1998).These characteristics are:

Protocol ID: The protocol associated with event. (TCP=0, UDP=1)

Source Port: The port number of the source.

Destination Port: The port number of the destination.

Source Address: The IP address of the source.

Destination Address: The IP address of the destination.

Raw Data Length: The length of the data in the packet.

Raw Data: The data portion of the packet.

Second level of the preprocessing involves converting one of the alphanumeric seven packet characteristics (Raw Data) into a numerical representation. This can be easily achieved by executing DISTINCT SELECT queries for the characteristic, loading results sets to a table and assigning unique integers to records in the result set.

Third level of preprocessing is selecting six of the original packet characteristics (Protocol ID, Source Port, Destination Port, Source Address, Destination Address, Raw Data Length) and the unique identifier of the remaining characteristic (Raw Data). In addition, an eight element (Attack), which was used to indicate whether the packet was an attack or not, was assigned to each record. A sample preprocessed input vector is shown in Table 4.1.

Table 4.1. Sample preprocessed input vector.

Protocol ID	Source Port	Destination Port	Source Address	Destination Address	Data ID	Raw Data Length	Attack
0	33694	161	2886758549	2886758500	1	2	1
0	33695	162	2886758549	2886758500	2	4	0
1	49345	69	3232235562	4294967295	1	2	0
1	49345	69	3232235562	4294967295	2	4	1

Preprocessed training and test data stored in a newly created table and a small program written in MATLAB was executed to extract data from database as array of vectors, which in turn would be used to train and test the neural networks used in the experiments.

4.3 Neural Network Structure

As there is no certain mathematical approach for obtaining the optimum number of hidden layers, two types of MLP networks, 3-layer MLP networks and 4-layer MLP networks, were employed in experiments. Generally neural networks with two hidden layers are capable of prediction and adding extra layers commonly yields similar results with two hidden layer networks, but their training periods are longer due to the more complex structures. Despite this fact, in addition to the 3-layer MLP networks 4-layer MLP networks were also utilized in experiments, in order to compare performances of 3-layer and 4-layer MLPs. For each of these types, two separate neural network sets were generated, one of which was trained with backpropagation algorithm with momentum while the other set was trained with the Levenberg-Marquardt algorithm. Eventually, we generated two sets of 3-layer and two sets of 4-layer MLP networks.

In addition to the MLP networks, four self-organizing maps were utilized in order to examine how well these networks can group the networks packets and whether the SOMs can be able to cluster similar attacks.

Details of MLP and SOM network structures used in experiments are given in section 4.3.1 and 4.3.2 respectively.

4.3.1. MLP Networks

MLP networks used to make basic input-output mapping. MLP networks would be trained in such way that, they would produce value of 1 if the presented input vector is attack and 0 if the presented input vector is normal network packet.

For all of the MLP networks, independent from the number of hidden layers, the number of input and output nodes is equal to the number of network inputs and desired outputs respectively. Therefore, input layer of all MLP networks contains seven nodes corresponding each network input while output layer contains only one node determining whether the presented input vector is an attack (with the value of 1) or not (with the value of 0). Hyperbolic tangent sigmoid activation function was used in all MLPs.

As there is no generic rule to find out the optimum number of hidden nodes needed, number of nodes in hidden layers is determined empirically for both 3-layer and 4-layer MLP networks.

For the set of 3-layer MLP networks, 25 different MLPs, which were formed by the combination of hidden layers with 5, 10, 15 20, 25 nodes, were generated. When a neural network is formed, weights between neurons are initialized randomly. As initial weight can significantly affect the performance of the neural network, each MLP was re-initialized ten times in order to avoid poor performance due to bad initial weights. Eventually, two set of 3-layer MLPs, each containing 250 MLP networks, were generated. One set was trained with backpropagation algorithm and the other set was trained with Levenberg-Marquardt algorithm.

For the set of 4-layer MLP networks, 125 different MLPs, which were formed by the combination of hidden layers with 5, 10, 15 20, 25 nodes, were generated. Because of the long training periods, re-initialization process was not conducted. Like 3-layer MLPs, two sets were generated, one of which was trained with backpropagation algorithm and the other set was trained with Levenberg-Marquardt algorithm.

After the training process, all MLPs were tested against a test dataset, which is not part of the training data set and MLP networks which gave the minimum false positive and false negative rates was kept.

4.3.2. Self-Organizing Maps

SOMs are used to classify data, which share similar characteristics. As mentioned in chapter two, only input vectors are used in training phase of the SOMs and SOMs are able to cluster similar data without giving them correct answers. In our research, we proposed to use SOMs in order to cluster normal and attack packets.

In our experiments we utilized four SOMs having grid architecture with 8x8 10x10, 15x15, 20x20 nodes in the feature layer.

4.4. Training the Neural Networks

In the training phase we have both input patterns and desired outputs related to each input vector. Aim of the training in MLP networks is minimizing output produced by the neural network and the desired output. In order to achieve this goal, weights are updated by carrying out certain steps (which is called as the training algorithm).

When using a supervised learning algorithm (e.g. backpropagation , Levenberg-Marquardt), training process is usually terminated when the RMSE is reduced to an acceptable level. There is no standard for the RMSE, but usually the lower it is, the better the classification rate is. But a too low RMSE may result in over training of the neural network. This means that neural network loose generalization ability, hence it will just detect attacks that are exactly identical to the training data.

Another criterion for training termination is the number of iterations. When a certain number of iterations were reached, the training was stopped, even if the desired RMSE was not reached.

250 3-layer and 125 4-layer MLP networks were trained with gradient descent backpropagation algorithm with a momentum rate of 0.5. Maximum number of epochs is 1000 and desired RMS was 0.0001.

250 3-layer and 125 4-layer MLP networks were trained with Levenberg-Marquardt algorithm. Maximum number of epochs is 50 and desired RMSE was 0.0001.

Training of the SOMS is different from MLP networks, as in the training phase of the SOMs desired outputs corresponding to the input vectors are not needed. But after the training process every node in the feature map should be labeled.

Training data set contains 10.421 input patterns, of which 971 are attacks.

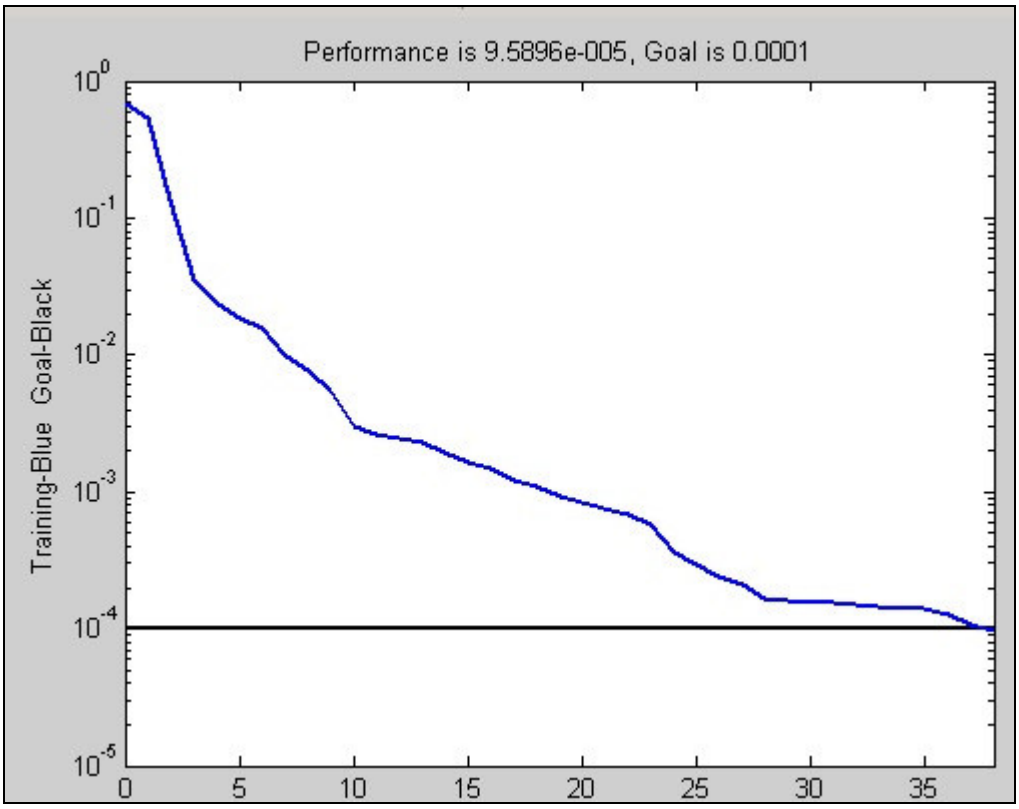

Figure 4.5. A sample training session where performance goal was met in 38 epochs.

4.5. Testing the Neural Networks

After the training is completed, the weights of the neural networks are frozen and performance of the neural networks evaluated. Testing the neural networks involves two steps, which are verification step and recall (or generalization) step.

In verification step, neural networks are tested against the data which are used in training. Aim of the verification step is to test how well trained neural networks learned the training patterns in the training dataset. If a neural network was trained successfully, outputs produced by the neural network would be similar to the actual outputs.

In recall or generalization step, testing is conducted with data which not used in training. Aim of the generalization step is to measure generalization ability of the trained network Recall data set contains 4306 input pattern, of which 73 are attacks.

CHAPTER 5

RESULTS AND DISCUSSION

This chapter presents the experimental results obtained by using the neural network based intrusion detection system, of which the methodology was already explained in the previous chapter. After the training process was completed, testing was conducted basically in two steps. In the first step, five sets of trained neural networks (3-layer backpropagation networks, 3-layer Levenberg-Marquardt networks, 4-layer backpropagation networks, 4-layer Levenberg-Marquardt networks and self-organizing maps) were tested against the training dataset, in order to examine how well neural networks 'learned' the training dataset after the training process. In the second step of the testing, trained neural networks were tested against a dataset, which is not a part of the training set, in order to examine generalization performance of the trained networks. In both testing steps performance of the neural networks was evaluated by examining the number of false positives and false negatives that they generated.

5.1. Verification Step

First step of the testing is the verification, where trained neural networks were tested against the training dataset. The aim of this step is to check if the neural networks were trained correctly. If the neural networks were trained correctly, neither false negatives nor false positives should be observed, at least false negative and false positive rates should be at an acceptable level. Training dataset contains 9450 normal and 971 attack instances.

5.1.1. 3-Layer Backpropagation Neural Networks

Number of false positives generated by each of the 3-layer backpropagation network after the verification step is given in Figure. 5.1. 10 neural networks were generated for each of the neural network architecture. Reason for generating 10 neural

networks for each of the neural network architecture is to avoid poor performances due to the bad initialization of the neural network weights.

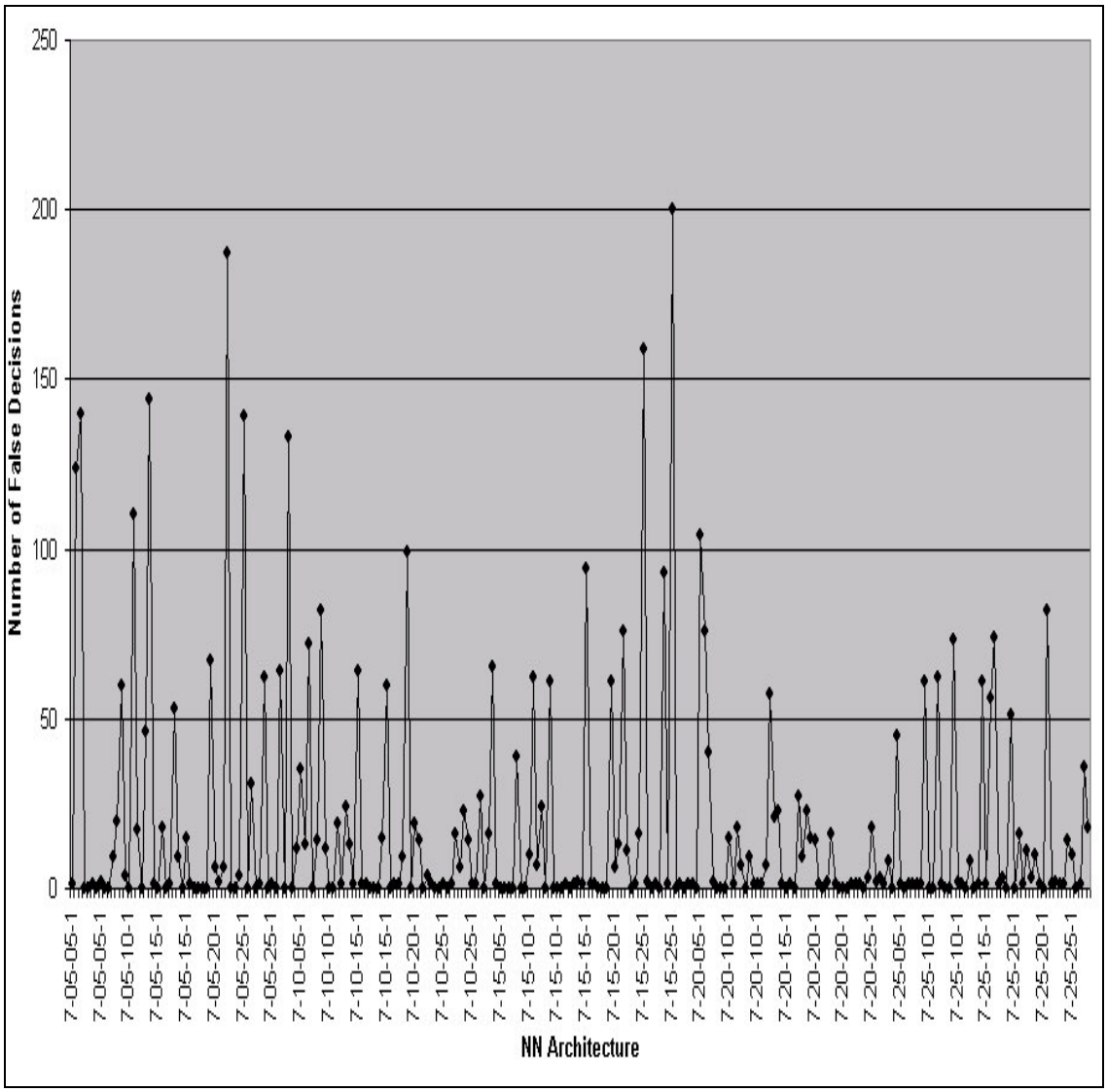

Figure 5.1. Number of false positives for 3-layer backpropagation networks.

From the Figure 5.1, it can be said that 3-layer backpropagation networks learned the normal network packets within the training dataset quite fairly, as 78 of 250 3-layer backpropagation networks generated no false positives, whereas the number of 3-layer backpropagation networks generating less than 10 false positives is 91. There are approximately 9800 normal network packets in the training dataset, and most of the 3-layer backpropagation networks recognized the normal network packets in the training dataset.

It can be also seen from Figure 5.1 that, initial values of the weights can significantly affect performance of the neural network, as neural networks that have the same architecture but different initial weight values show different performances.

Another point which should be mentioned is that, increasing number of hidden nodes would not improve the performance. For instance, most of the neural networks having 7-5-5-1 architecture show better performances than the most of the neural networks having 7-15 -25-1 architecture.

Another criterion used to determine the performance of the neural networks is the number of false negatives, which is the number of the missed attacks. Number of the false negatives encountered by the 3-layer backpropagation networks is given in Figure 5.2.

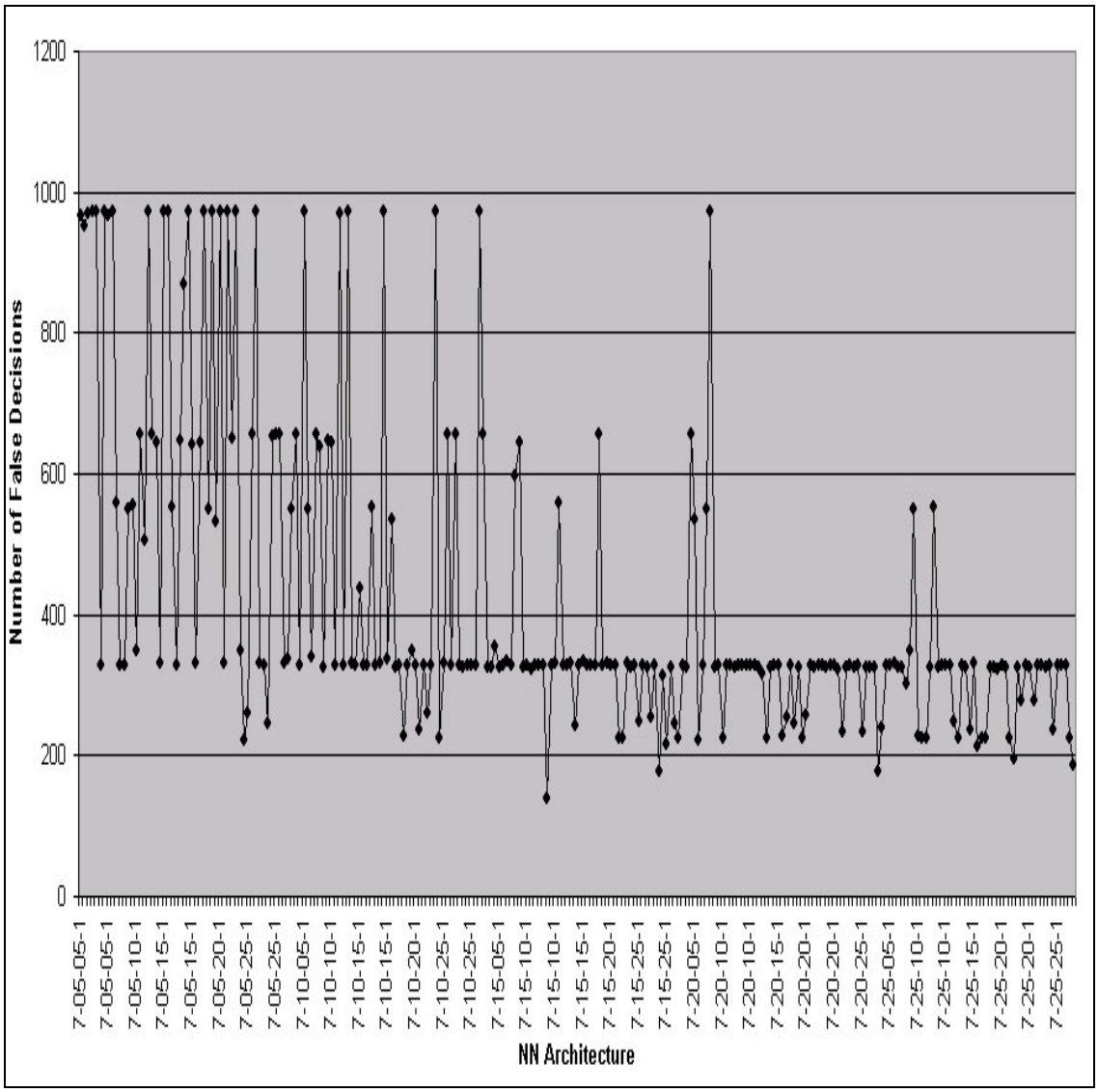

Figure 5.2. Number of false negatives for 3-layer backpropagation networks.

Number of the false negatives encountered by the 3-layer backpropagation networks is surprising. Although neural networks learned the normal packets in the training dataset quite fairly, they failed to distinguish attack packets in the training dataset. There are 971 attack instances in the training dataset, and the neural network that showed best performance by mean of false negatives is the one having 7-15-25-1 architecture and an identity of 6 with 176 false negatives.

5.1.2. 3-Layer Levenberg-Marquardt Neural Networks

Number of false positives generated by each of the 3-layer Levenberg-Marquardt network after the verification step is given in Figure 5.3.

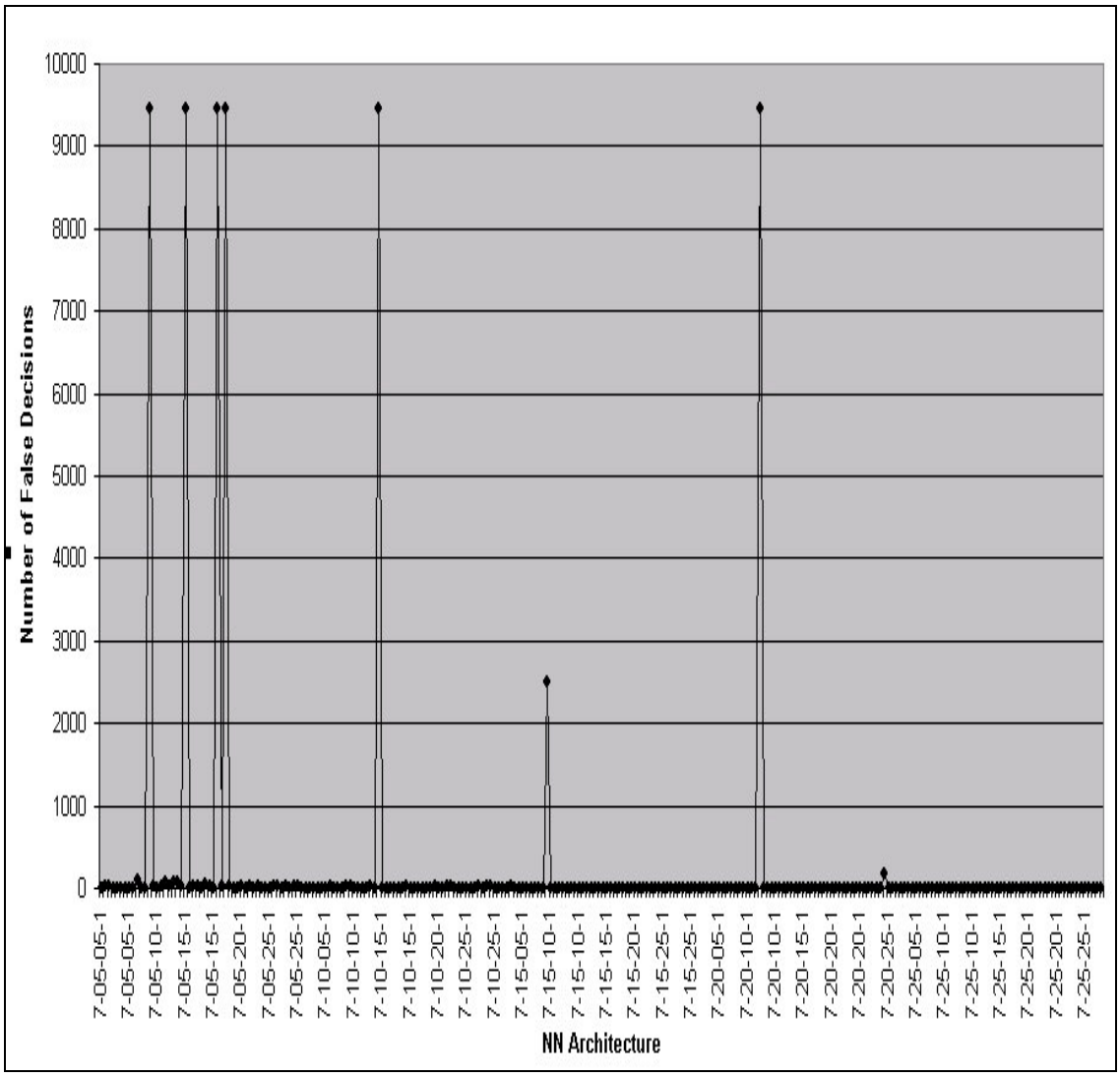

Figure 5.3. Number of false positives for 3-layer Levenberg-Marquardt networks.

After examining Figure 5.3, it can be said that with some exceptions 3-layer Levenberg-Marquardt networks learned the normal packets in the training dataset. Number of 3-layer Levenberg-Marquardt networks, which generated no false positives, is 92. In addition, 111 of 250 3-layer Levenberg-Marquardt networks generated less than 10 false positives. Except for the neural networks having 5 hidden nodes in the first hidden layer, all of the other neural networks successfully learned the normal packets in the training dataset. In addition, it was observed that, some networks (for instance neural network having 7-20-10-1 architecture and an identity of 5) showed poor performances, probably due to the bad initialization of the weights. Neural networks having 25 hidden nodes in the first hidden layer showed the best performances among the other neural networks having different architecture.

Number of the false negatives encountered by the 3-layer Levenberg-Marquardt networks is given in Figure 5.4.

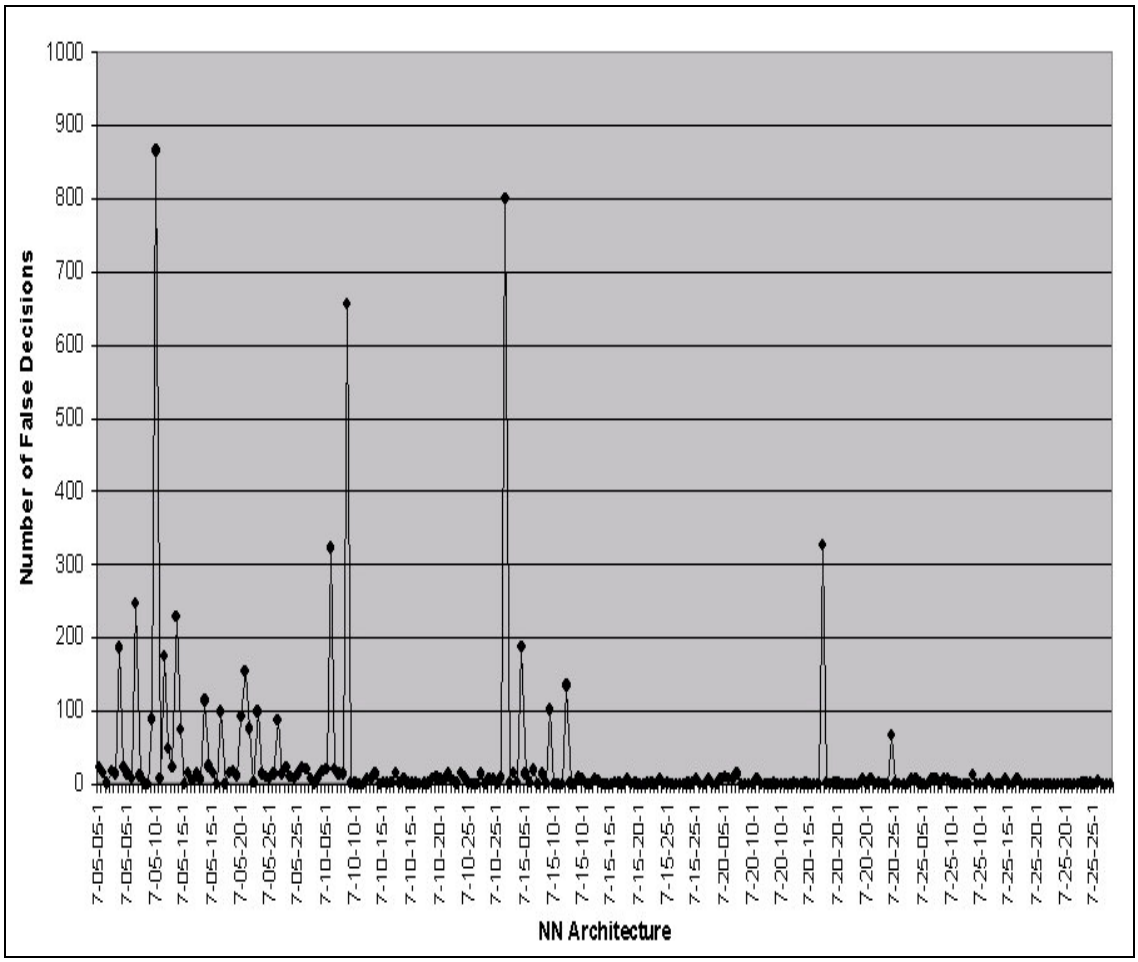

Figure 5.4. Number of false negatives for 3-layer Levenberg-Marquardt networks.

False negative results obtained by the 3-layer Levenberg-Marquardt networks are promising. Unlike from the 3-layer backpropagation networks, most of the 3-layer Levenberg-Marquardt networks successfully distinguished attacks instances in the training dataset. 98 of 250 3-layer Levenberg-Marquardt networks generated no false negatives. Beyond some instances of the other network architectures, nearly all neural networks having 20 or 25 hidden nodes in the first hidden layer produced no false negatives. 92 of 250 3-layer Levenberg-Marquardt networks generated less than 10 false negatives.

5.1.3. 4-Layer Backpropagation Neural Networks

Number of false positives generated by each of the 4-layer backpropagation network after the verification step is given in Figure 5.5. As managing whole neural networks would be a troublesome task and training periods would take long, 4-layer backpropagation networks were not re-initialized.

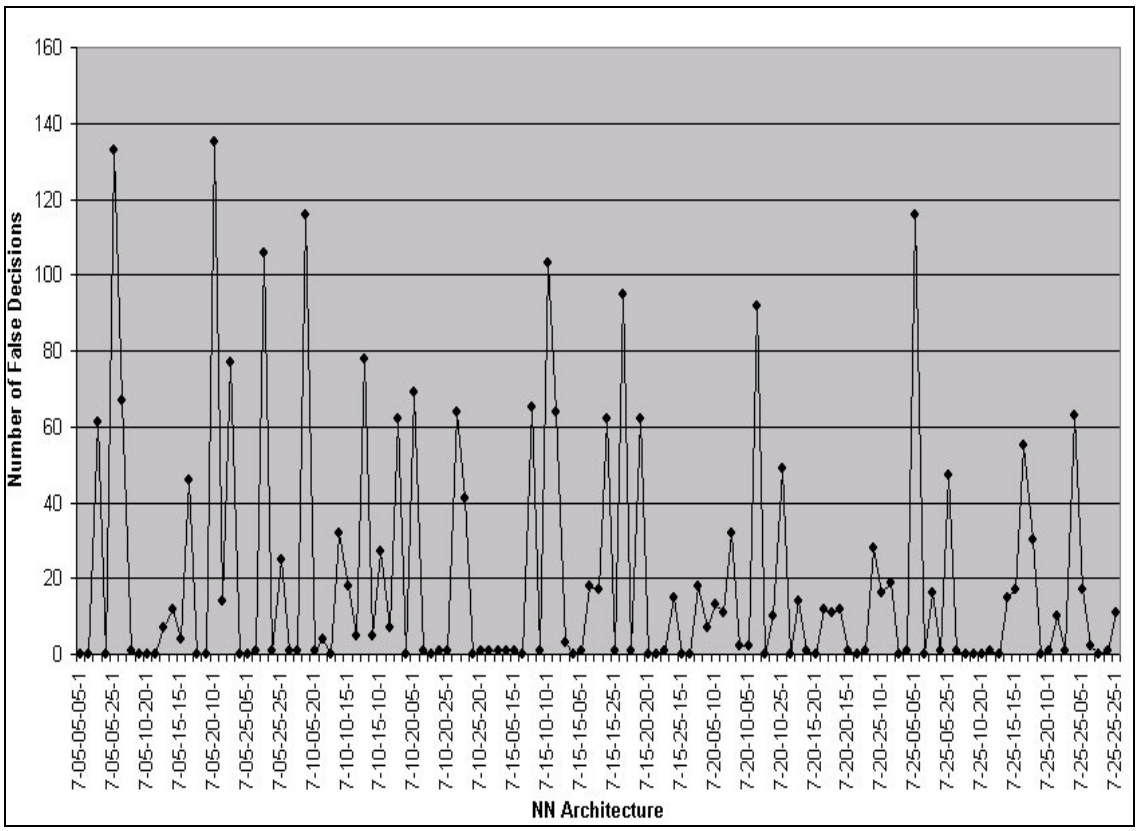

Figure 5.5. Number of false positives for 4-layer backpropagation networks.

From Figure 5.5, it can be said that 4-layer backpropagation networks can fairly distinguish normal network packets in the training dataset. 32 of 125 4-layer backpropagation networks successfully identified normal packets in the training dataset without generating any false positives. Additionally, 74 of 125 4-layer Levenberg-Marquardt networks generated less than 10 false positives.

Number of false positives encountered after testing 4-layer backpropagation networks against the training dataset is given in Figure 5.6.

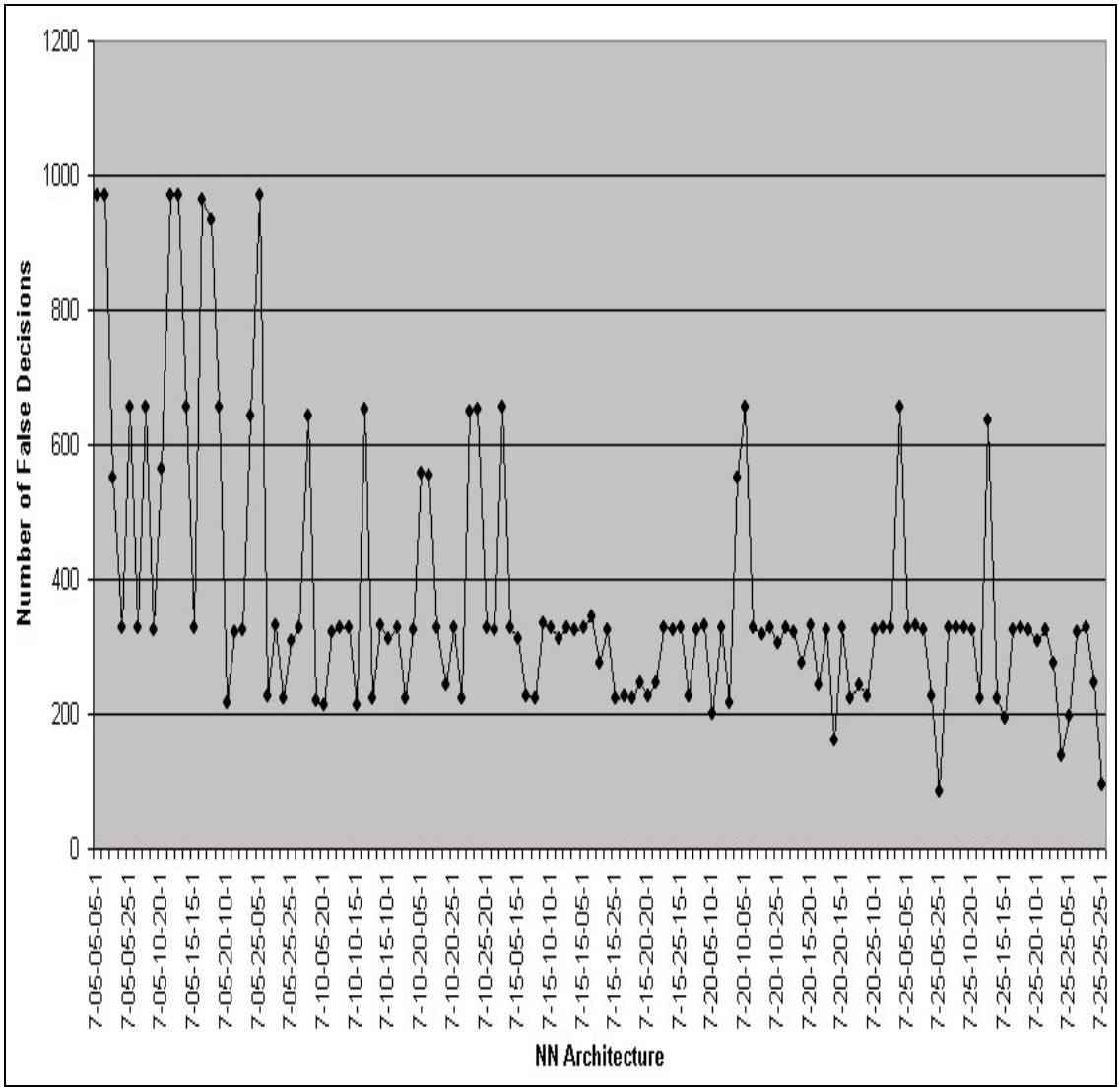

Figure 5.6. Number of false negatives for 4-layer backpropagation networks.

It is observed that, like 3-layer backpropagation networks, 4-layer backpropagation networks failed to learn the attack instances in the training dataset.

Number of false negatives is relatively high with respect to the 971 attack instances in the training dataset. Neural network with architecture of 7-25-05-25-1 showed the best performance with 86 false negatives.

5.1.4. 4-Layer Levenberg-Marquardt Neural Networks

Numbers of false positives generated by 4-layer Levenberg-Marquardt networks after presenting training dataset is given in Figure 5.7.

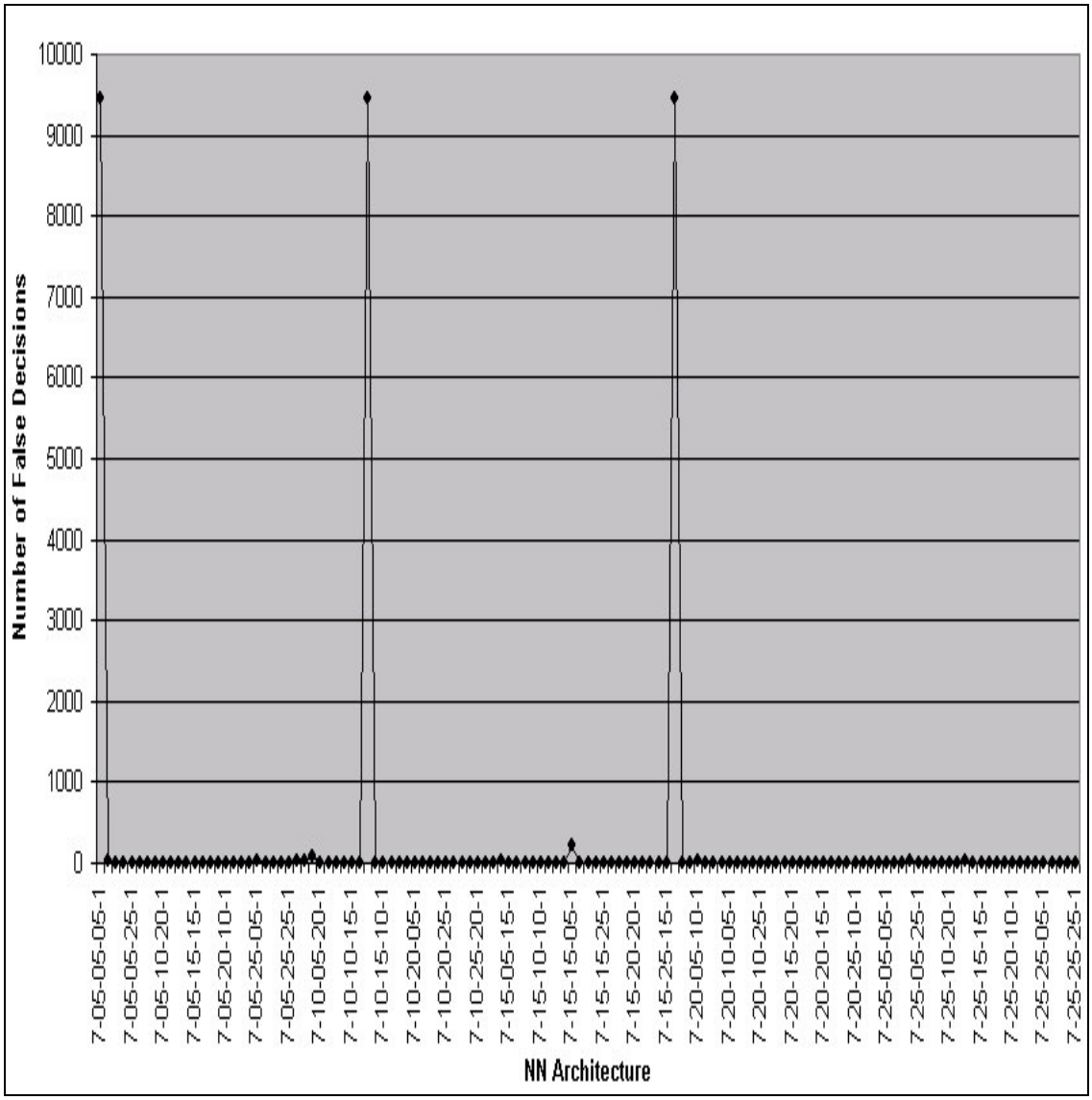

Figure 5.7. Number of false positives for 4-Layer Levenberg-Marquardt networks.

It can be seen from the Figure 5.7, most of the 4-layer Levenberg-Marquardt networks could successfully recognize normal attack packets in the training dataset. 79 of 125 4-layer Levenberg-Marquardt networks successfully identified normal packets in the training dataset without generating any false positive. 31 of 125 4-layer Levenberg-Marquardt networks generated less than 10 false negatives. However, 7-5-5-5-1 and 7-25-20-15-1 networks failed to recognize normal packets in the training dataset and flagged all normal packets as attack.

Number of false negatives generated by 4-layer Levenberg-Marquardt networks is given in Figure 5.8.

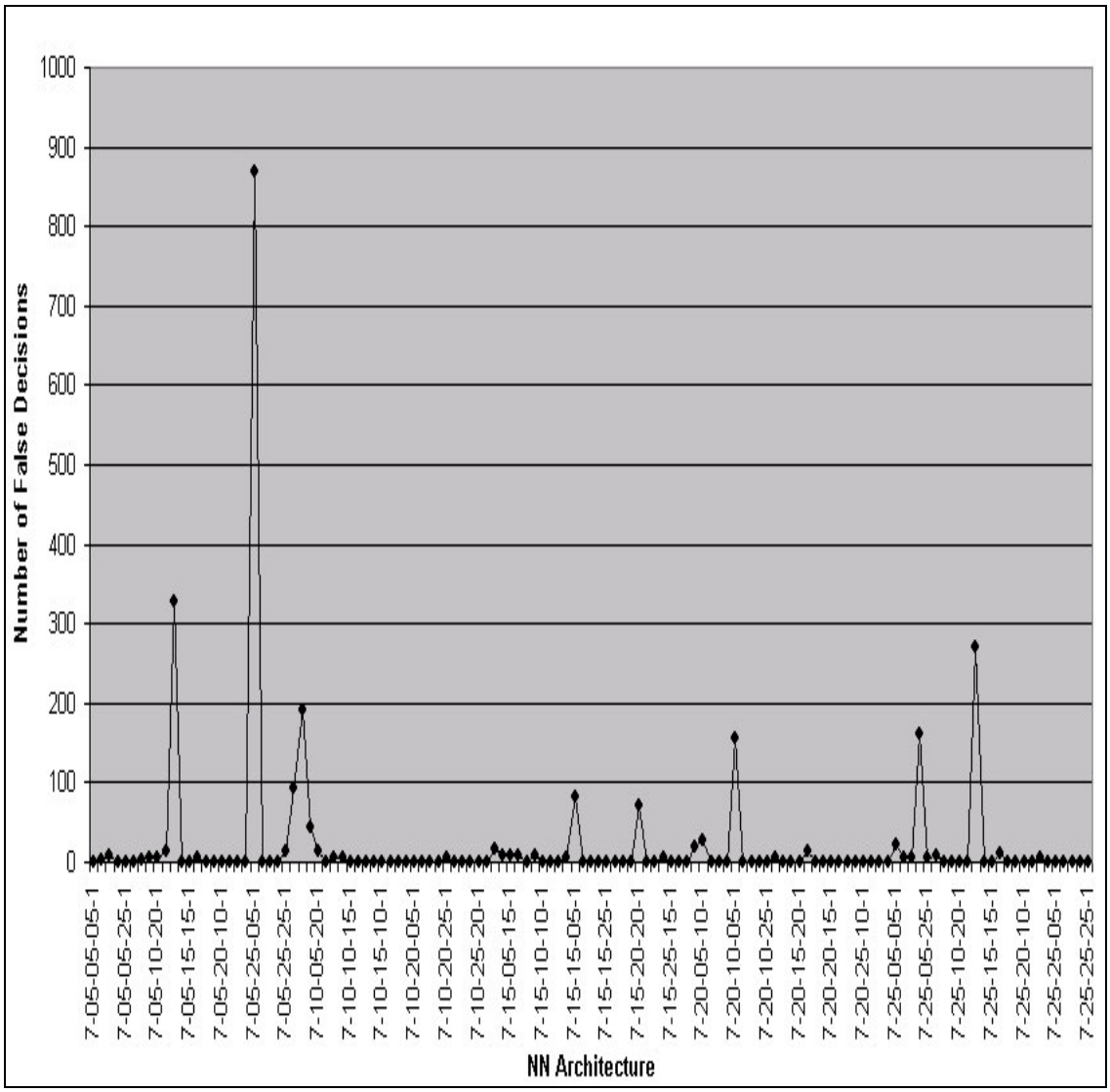

Figure 5.8. Number of false negatives for 4-layer Levenberg-Marquardt networks.

All of the 4-layer Levenberg-Marquardt networks but neural networks of 7-5-10-10-1, 7-5-20-25-1, 7-10-5-20-1, 7-15-5-5-1, 7-15-5-25-1, 7-25-5-5-1 could learn the attack packets in the training dataset. 60 of 125 4-layer Levenberg-Marquardt networks generated no false negatives when they were tested against the training dataset. 46 of 125 4-layer Levenberg-Marquardt networks generated less than 10 false negatives.

5.1.5. SOMs

In our study, four types of SOMs with 8x8, 10x10, 15x15 and 20x20 nodes in the feature layer were trained. The aim of using SOMs in this research is to train SOMs such that every node in the feature layer would represent an attack type. In verification step, every training example is presented to the SOMs and which training example activates which output node in the feature map was inspected. By doing so every output node in the feature map was labeled.

Although our aim for using SOMs in the field of intrusion detection is to classify attacks, at the end of the verification step, it is observed that all of the four SOMs failed to classify attack instances in the training dataset, but instead it was seen that similar attacks were grouped together. As a result, SOMs were used to distinguish normal and attack packets instead of classifying attacks.

For the 8x8 SOM, attacks in the training dataset were distributed among the output nodes of 35, 42 56 57 58 64. 8x8 SOM gave no false negatives in verification step, but 9450 of normal packets, 3979 of them were labeled incorrectly as attack.

For the 10x10 SOM, attacks in the training dataset were distributed among the output nodes of 9, 10, 55, 65, 66, 67, 74 and 97. 10x10 SOM gave also no false negatives, but 9450 of normal packets, 1127 of them were labeled incorrectly as attack.

For the 15x15 SOM, attacks in the training dataset were distributed among the output nodes of 35, 78, 95, 96, 110, 11, 179, 180, 194, 209 and 225. 15x15 SOM gave no false negatives, but 1768 of the normal packets were labeled incorrectly as attack.

For the 20x20 SOM, attacks in the training dataset were distributed among the output nodes of 2, 91, 92, 108, 109, 110, 11, 130, 131, 149, 151, 361, 362, 363 and 381. 20x20 SOM gave also no false negatives, but 750 of the normal packets were labeled incorrectly as attack.

5.2. Generalization Step

In the generalization step trained networks were tested against a dataset, which is not part of the training dataset. Aim of this step is to see how well trained networks would distinguish unknown normal and attack packets in the test dataset. In this step generalization ability of the trained networks would be evaluated, as trained network would decide whether an unknown packet is an attack or not by generalizing known packets learned in training process. There are 4233 normal and 73 attack packets in the test dataset.

5.2.1. 3-Layer Backpropagation Neural Networks

Number of false positives generated by each of the 3-layer backpropagation network after the verification step is given in Figure 5.9.

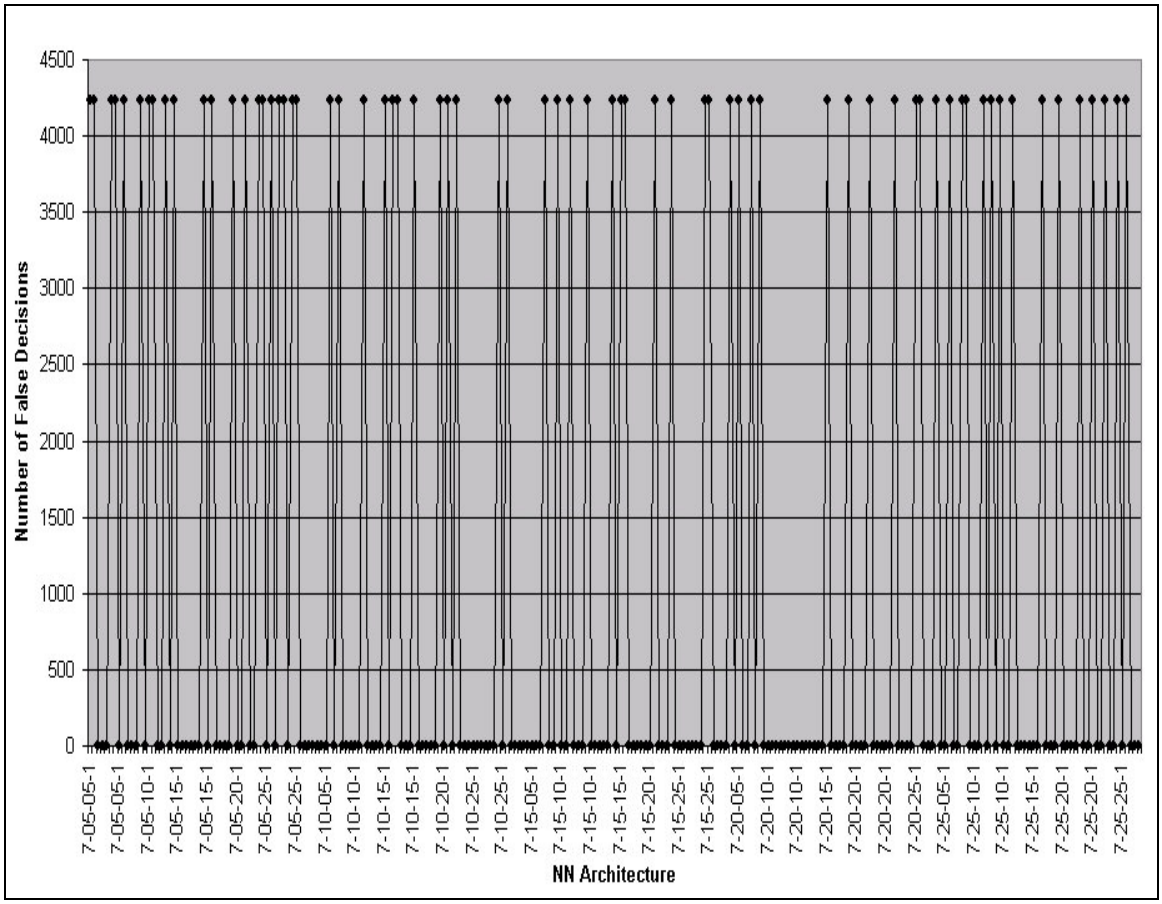

Figure 5.9. Number of false positives for 3-layer backpropagation networks.

It is observed from the Figure 5.9 that, of the 250 trained 3-layer backpropagation networks, 181 of them could successfully generalize test dataset from the training dataset. Rest of the 69 3-layer backpropagation networks were not trained correctly and they flagged all normal network packets as attack.

Number of the false negatives encountered by the 3-layer backpropagation networks is given in Figure 5.10.

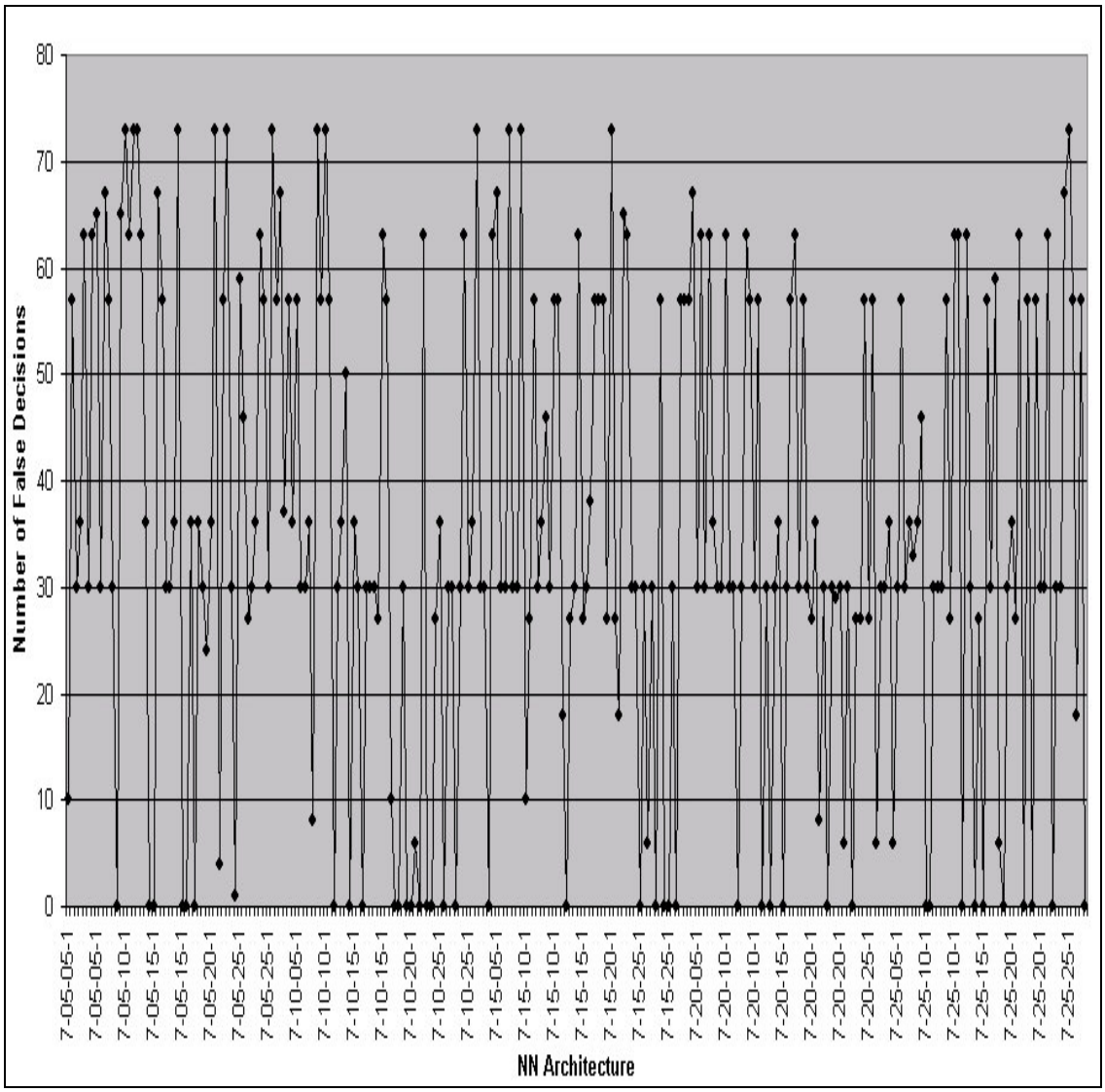

Figure 5.10. Number of false negatives for 3-layer backpropagation networks.

It can be seen from the Figure 5.10, most of the 3-layer backpropagation failed to catch attacks in the test dataset. Only a few numbers of 3-layer backpropagation

networks, namely, 1st and 7th networks of 7-5-10-1 architecture, 5th network of the 7-15-15-1 architecture, 10th network of the 7-15-20-1 architecture, 10th network of the 7-15-25-1 architecture,7th network of the 7-20-5-1 architecture, 10th network of the 7-20-25-1 architecture and 10th network of the 7-25-20-1 architecture, could successfully identify attacks in the test dataset with no false negatives.

5.2.2. 3-Layer Levenberg-Marquardt Neural Networks

Number of false positives generated by each of the 3-layer Levenberg-Marquardt network after the verification step is given in Figure 5.11.

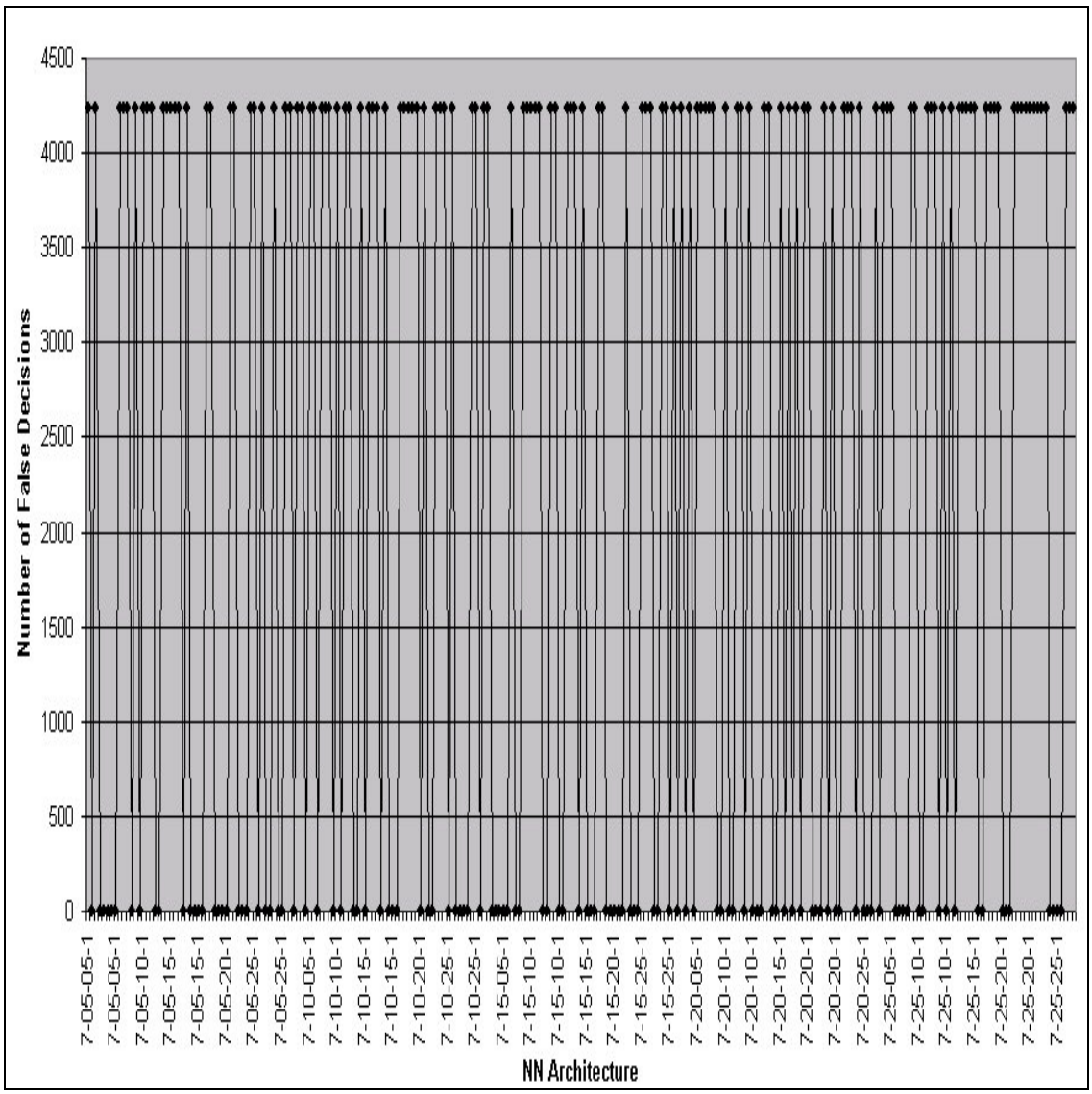

Figure 5.11. Number of false positives for 3-layer Levenberg-Marquardt networks.

119 of the 250 3-alyer Levenberg-Marquardt networks could successfully identify normal packets in the test dataset without giving any false positives. The rest of the 131 3-layer Levenberg-Marquardt networks failed to recognize normal packets in the test dataset, as they labeled all normal packets as attack.

Number of the false negatives encountered by the 3-layer Levenberg-Marquardt networks is given in Figure 5.12.

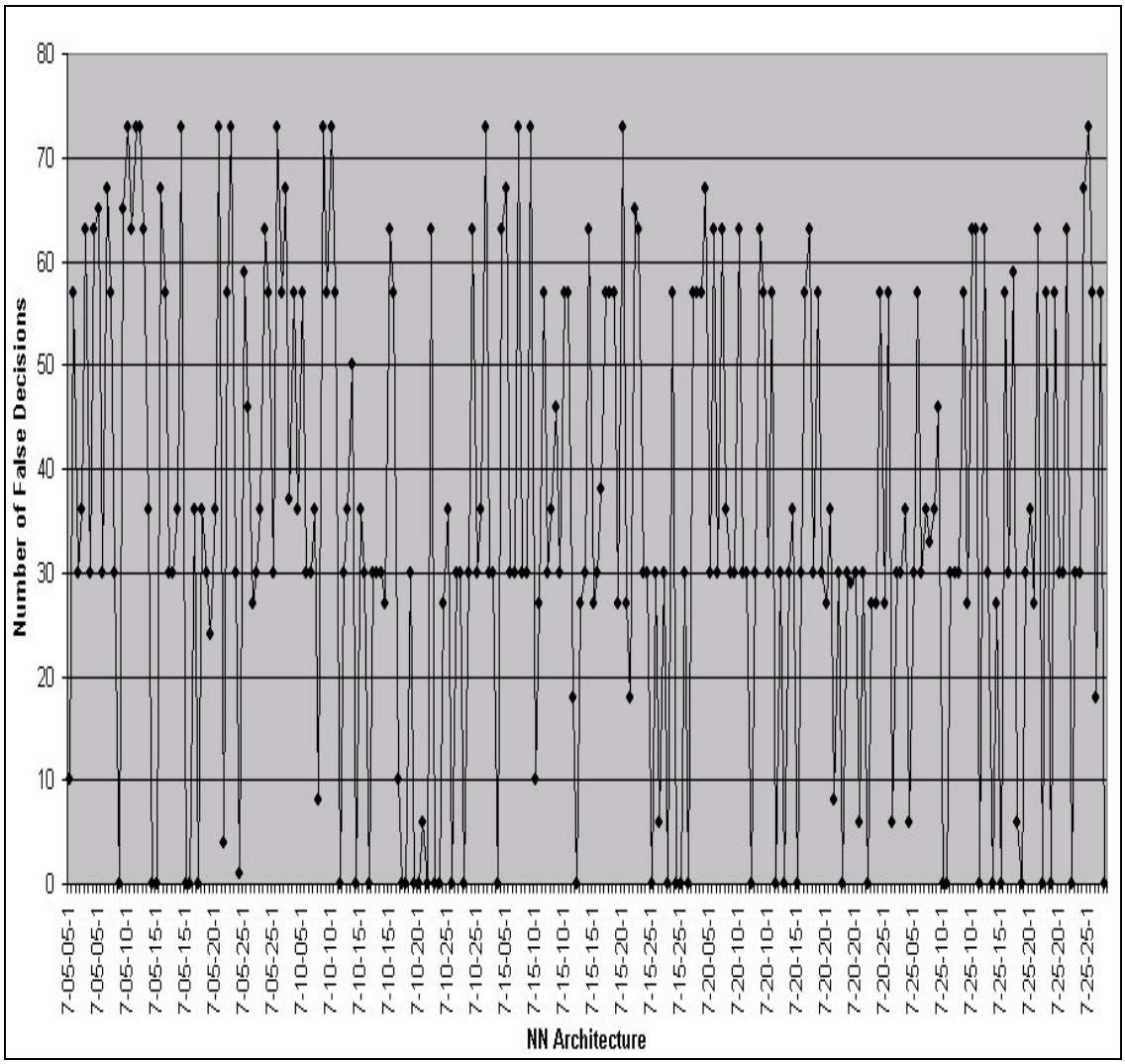

Figure 5.12. Number of false negatives for 3-layer Levenberg-Marquardt networks.

It can be seen from Figure 5.12, 41 of 250 3-layer Levenberg-Marquardt networks, which have different architectures, successfully recognized the attack patterns in the test dataset without generating any false negatives.

5.2.3. 4-Layer Backpropagation Neural Networks

Number of false positives generated when test dataset was presented to the 4-layer backpropagation networks is given in Figure 5.13.

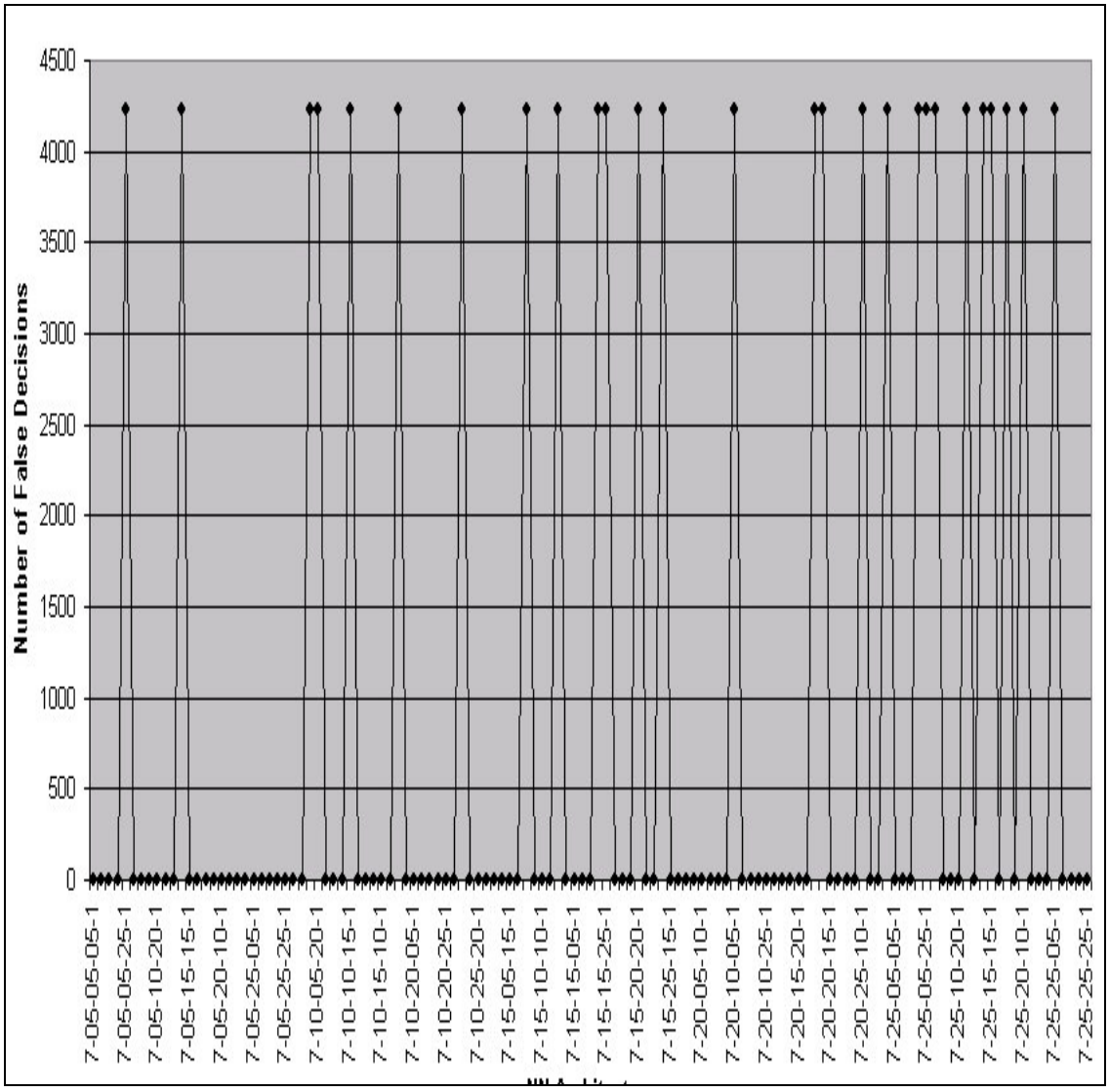

Figure 5.13. Number of false positives for 4-layer backpropagation networks.

Results show us that, 98 of 250 4-layer backpropagation networks could successfully identify normal packets in the dataset without giving any false positives. But performances of the rest of 152 4-layer backpropagation networks are poor, as all of them flagged normal packets in the test dataset as attack.

Number of false negatives generated when test dataset presented to the 4-layer backpropagation networks is given in Figure 5.14.

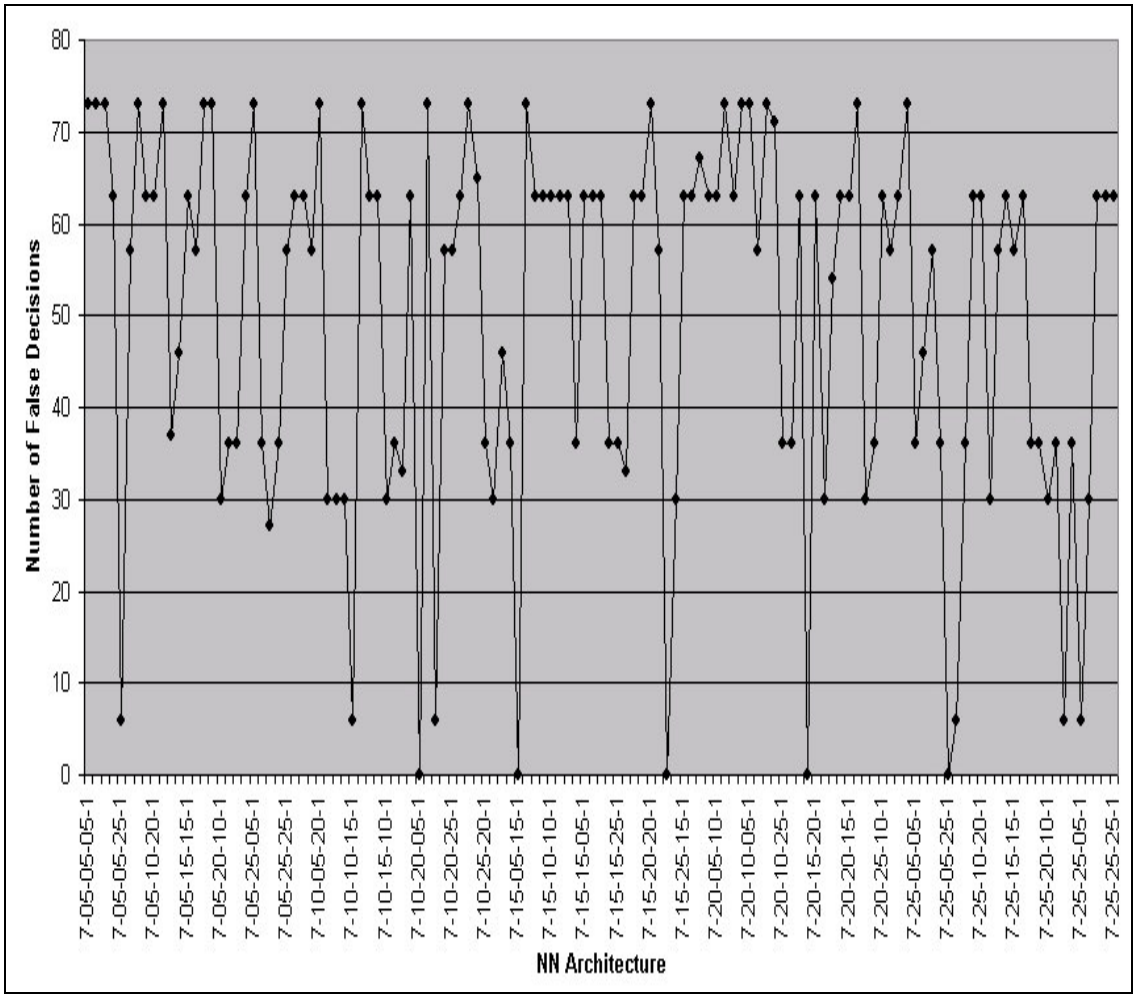

Figure 5.14. Number of false negatives for 4-layer backpropagation networks.

Unlike from the performance in false positive rates, most of the 4-layer backpropagation networks showed poor performances in the context of number of false negatives. Only five 4-layer backpropagation networks, namely neural networks having the architectures of 7-10-20-05-1, 7-15-05-15-1, 7-15-25-05-1, 7-20-15-15-1, 7-25-05-25-1, could successfully detect all attacks in the test dataset.

5.2.4. 4-Layer Levenberg-Marquardt Neural Networks

Number of false positives generated when test dataset is presented to the 4-layer Levenberg-Marquardt algorithm is given in Figure 5.15.

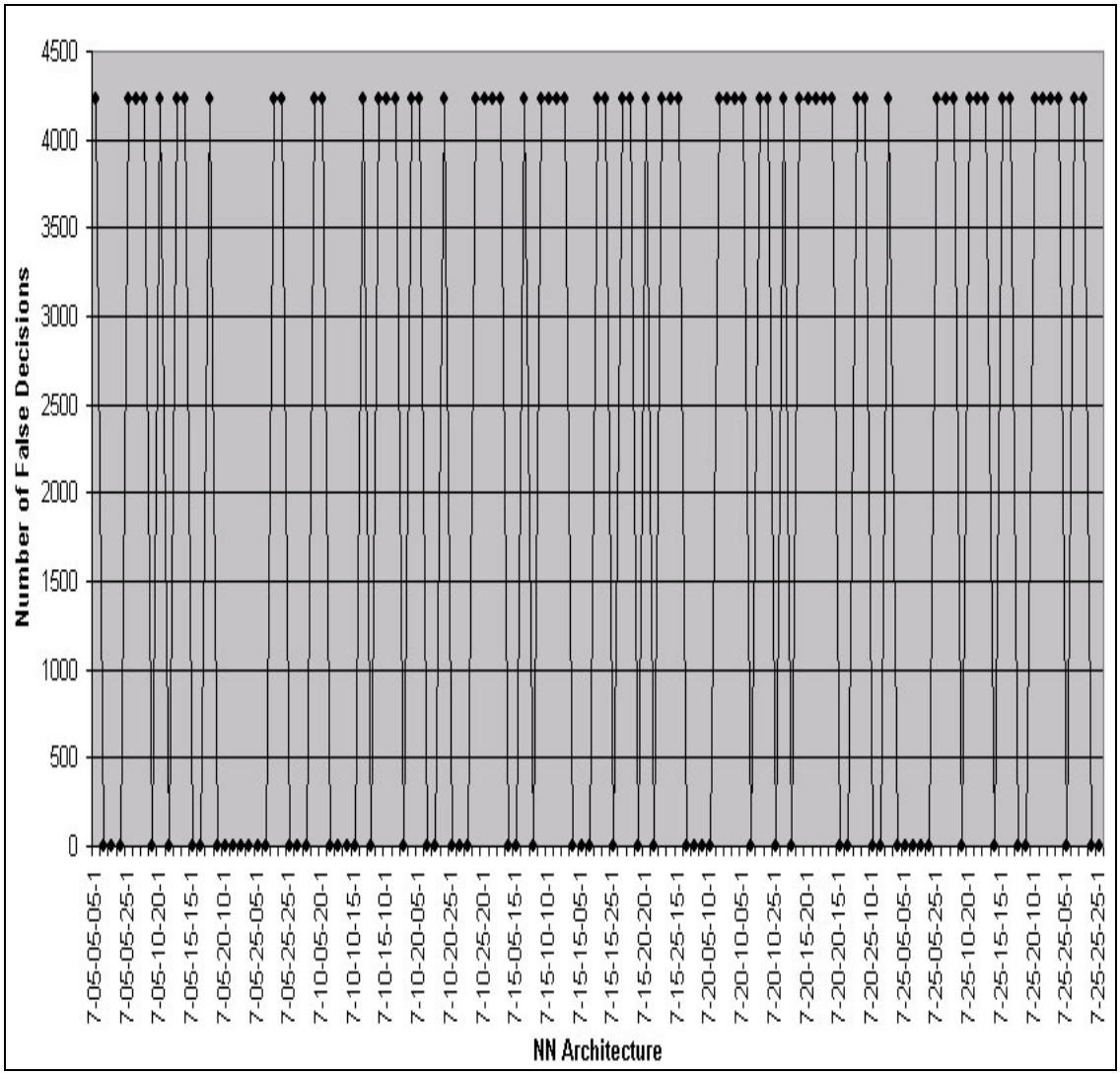

Figure 5.15. Number of false positives for 4-layer Levenberg-Marquardt networks.

Examination of the Figure 5.15 showed us that, 60 of 250 networks could successfully identify normal packets in the test dataset without generating any false positives. Rest of the 165 networks labeled all normal packets in the test dataset as attack.

Number of false negatives encountered by 4-layer Levenberg-Marquardt networks when test dataset is presented is given in Figure 5.16.

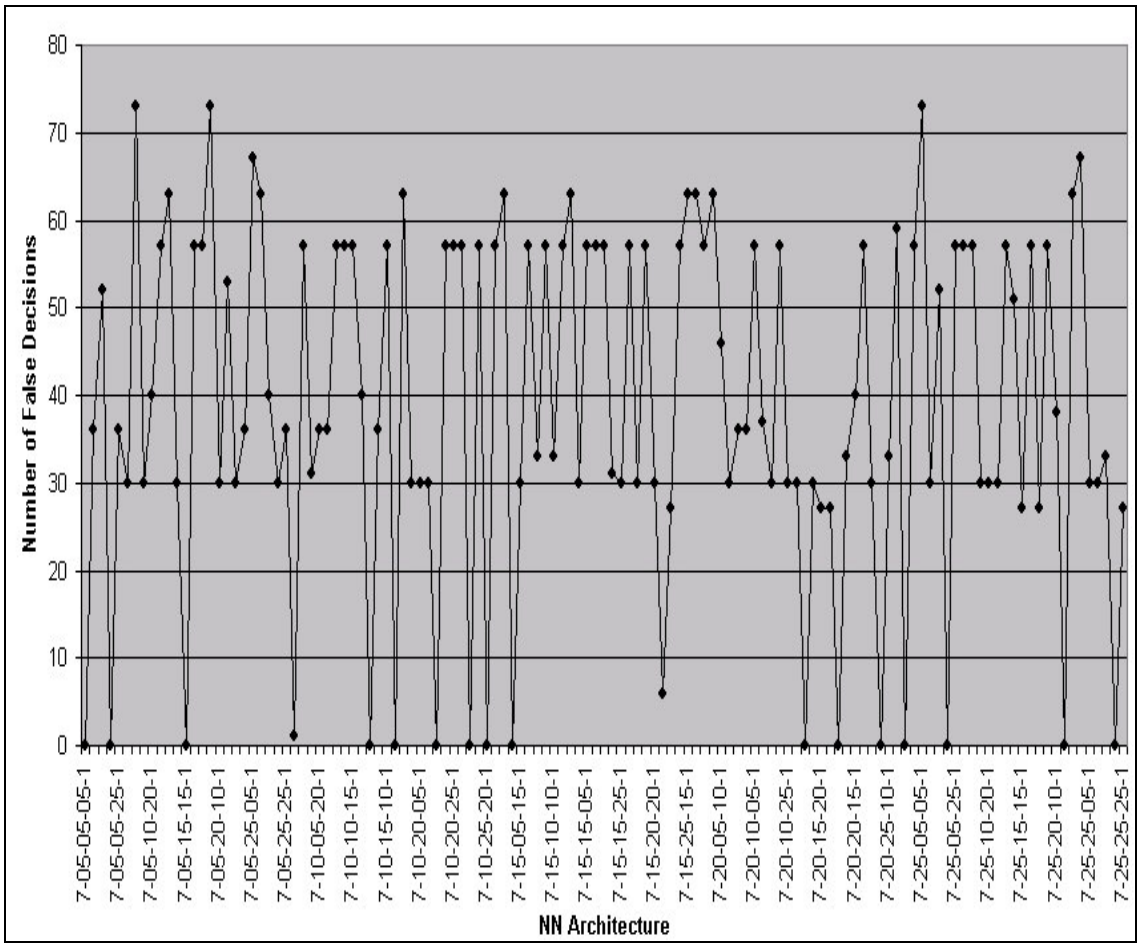

Figure 5.16. Number of false negatives for 4-layer Levenberg-Marquardt networks.

Results of the experiments showed us that, 16 of 125 4-layer Levenberg-Marquardt networks could successfully distinguish attack packets in the test dataset without giving any false negatives.

5.2.5. SOMs

In the generalization step, in the context of false negatives all of the four SOMs performed well, as none of them generated false negatives. Number of false positives generated among 4233 normal packets, SOMs having 8x8, 10x10, 15x15, 20x20 nodes in feature layer generated 1845, 819, 830 and 332 false positives respectively.

5.3. Analysis

Although many trained neural networks could successfully distinguish attack and normal packets in the either training dataset or test dataset, number of trained networks, which could successfully distinguish normal and attack packets in both training and test dataset, is few.

None of the trained 3-layer and 4-layer backpropagation networks could successfully identify normal and attack instances in both of the training dataset and test dataset. Although some of them successfully identify attacks and normal packets in training dataset and some of them successfully identify attack and normal packets in the test dataset set, none of them achieved to identify all attack and normal packets in both of the training and test dataset.

Similar results were obtained for 3-layer and 4-layer Levenberg-Marquardt networks. Like backpropagation networks, many of the Levenberg-Marquardt networks could successfully distinguish attack and normal packets either in training dataset or in test dataset. However, number of Levenberg-Marquardt networks, which could successfully distinguish attack and normal packets in both training and test dataset, is too low. Among 250 3-layer Levenberg-Marquardt networks, number of neural networks, which could successfully identify all attacks in both datasets, is four. These networks are: 6^{th} network of the 7-10-25-1 architecture, 5^{th} network of the 7-15-25-1 architecture, 9^{th} network of the 7-20-25-1 architecture, 5^{th} network of the 7-20-20-1 architecture. There are five 4-layer Levenberg-Marquardt networks, which could successfully distinguish attack and normal packets both in training dataset and test dataset. These networks are neural networks having the architectures of 7-15-15-5-1, 7-15-10-20-1, 7-25-20-20-1, 7-25-20-25-1 and 7-20-25-15-1.

Results of the SOMs' experiments showed us that, although all SOMs failed to classify attack types in the training dataset, they successfully identified attacks from normal packets in training and test datasets. It was observed that, attacks in the datasets were distributed among few output nodes. All of the SOMs could successfully identify attacks without generating any false negatives in both of the two test steps. The SOM having 20x20 nodes in feature map outperformed other SOMs having different architectures with generating 750 and 332 false positives when tested against training and test dataset respectively.

5.4. Discussion

After the examination of the experimental results, it is observed that many networks failed to distinguish attack and normal packets in training dataset, which is in turn resulted in high false positive and false negative rates. This is due to the fact that, these networks could not achieve to reach desired error rate of 0.0001 in given iterations (1000 for backpropagation algorithm and 50 for Levenberg-Marquardt algorithm), hence they failed to learn normal and/or attack packets in the training datasets. As these networks could not be trained correctly, they also failed to distinguish normal and attack packets in the test dataset.

It is also observed that, although some neural networks were trained correctly and they could successfully recognize normal and attack packets in the training dataset, they failed to distinguish attack and/or normal packets in the test dataset. This is due to the fact that, these networks were over-trained; hence they could not generalize packets in test dataset from learned packets in training dataset.

It is seen that, assignment of initial weight values can significantly affect the performance of the neural networks, as neural networks having same architectures showed different performances due to the different initial weight values.

It is worth to mention that, we found out increasing the number of hidden layers may not cause an increase in performance, as four 3-layer Levenberg-Marquardt networks achieved to distinguish attack and normal packets in both training and test dataset. But we experienced that, training periods of the 4-layer networks are longer than the 3-layer networks due to their more complex structures.

None of the backpropagation networks could successfully identify attacks and/or normal packets in both tests, while nine Levenberg-Marquardt networks could successfully distinguishes normal and attack packets in both tests. This is due to the fact that, Levenberg-Marquardt training algorithm is faster than backpropagation algorithm and Levenberg-Marquardt networks were trained more accurately.

All trained SOMs could successfully identify attack instances in training and test dataset. It was observed that, increasing number of nodes in feature map improves false positive rates.

CHAPTER 6

CONCLUSION

In this study a network-based neural network intrusion detection system was presented. The main problem with current IDSs is that they produce many false alarms. In addition to that, writing rules or signatures is not an easy task and can be time consuming. The purpose of this study was to develop a neural network based intrusion detection system in order to decrease false positive and false negative rates as neural networks can generalize unknown network packets from known ones. Additionally, with using neural networks writing rules for every attack and regular updating of the attack signatures can be avoided.

Two training algorithms; backpropagation and Levenberg-Marquardt algorithm, were used in the training of neural networks. For each of these training algorithms a 3-layer and a 4-layer MLP network sets were generated. As there is no rule for determining the optimum number of nodes in hidden layer, variety of MLP networks, each of which has different number of nodes in hidden layer(s), were generated and the MLP networks, which performed best in testing step, was kept and the others were discarded.

Tests were conducted in two steps. In the first step trained networks were tested against the training dataset to see how well trained MLP networks learned the training set. In the second step of the testing, trained MLP networks tested against a dataset, which is not part of the training dataset, to see generalization ability of the trained networks.

Analysis of the experimental results show us that, both 3-layer and 4-layer backpropagation networks failed to distinguish normal and attack packets either in training dataset or in test dataset. This is due to the fact that, some of the backpropagation networks could not be trained correctly and showed poor performances in the test against training dataset whereas some of them were over-trained, hence could not generalize unknown packets from known ones and showed poor performances in the test against test dataset. We also observed that, a few number of MLP networks of 3-

layer and 4-layer Levenberg-Marquardt networks achieved to identify normal and attack packets in training and test dataset.

SOMs failed to classify attacks in the experiments, but they performed well when identified attack instances in training and test datasets.

Nine of the trained Levenberg-Marquardt networks gave no false positives and false negatives and outperformed similar studies which were summarized in Table 3.1.

In study, it is observed that, appropriately trained neural networks can learn intrusive and non-intrusive network packets and can distinguish normal and attack packets in an unseen dataset.

As future work, different characteristics of the network packet may be selected in addition to the seven characteristics. Additionally, new attacks may be added to the training dataset.

REFERENCES

Anderson, J. 1980. "Computer Security Threat Monitoring and Surveillance", James P. Anderson Co., Fort Washington, PA.

Bace, R. and Mell, P. 2001. “Intrusion Detection Systems”, NIST Special Publication, Nov 2001.

Bivens, A., Palagiri, C., Smith, R., Szymanski, B. and Emrechts, M. 2002. “Network-Based Intrusion Detection Using Neural Networks”, Intelligent Engineering Systems through Artificial Neural Networks, Vol. 12, Proc. ANNIE.

Cannady, J. 1998. “Artificial Neural Networks for Misuse Detection”, Proceedings, National Information Systems Security Conference (NISSC’98), October, Arlington, VA, pp. 443-456.

Ghosh, A. and Schwartzbard, A.1999. “A Study in Using Neural Networks for Anomaly and Misuse Detection”, USENIX Security Symp, Washington, D.C, USA, (23-26 August 1999).

Ghosh, A. K., Schwartzbard, A. and Schatz M. 1999. “Learning Program Behavior Profiles for Intrusion Detection”, Proceedings of the Workshop on Intrusion Detection and Network Monitoring, pp. 51-62.

Hagan, M.T. and Menhaj, M.B. 1994. "Training Feedforward Networks with the Marquardt Algorithm", IEEE Transactions on Neural Networks, Vol. 5, No 6, pp. 989-993.

Haines, J. W., Lippmann, R. P., Fried, D. J., Zissman, M. A., Tran E. and Boswell, S. B. 2001. “1999 DARPA Intrusion Detection Evaluation: Design and Procedures”, MIT Lincoln Laboratory Technical Report, TR-1062, Lexington, MA.

Haykin, S. 1999. “Neural Networks: A Comprehensive Foundation”, (Prentice Hall, New Jersey), pp. 161-167.

Jain, A. K., Mao, J. and Mohiuddin, K. 1996. “Artificial Neural Network: A Tutorial”, *IEEE Computer*, Vol. 29, No. 2, pp. 31-44.

Jirapummin, C., Wattanapongsakorn, N. and Kanthamanon P. 2002. “Hybrid Neural Networks for Intrusion Detection System”, The 2002 International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC 2002), pages 928-931, Phuket, Thailand, (16-19 July 2002).

Kasabov, N. K., 1996. “Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering”, (The MIT Press, Cambridge), p. 300.

Lee, S.C. and Heinbuch, D.V. 2001. "Training a Neural-Network Based Intrusion Detector to Recognize Novel Attacks", IEEE Trans. on Systems, Man, and Cybernetics, Part A, 31, pp. 294-299.

Lippmann, R. P. and Cunningham R. K. 2000. "Improving Intrusion Detection Performance Using Keyword Selection and Neural Networks", Computer Networks, Vol. 34, pp. 597-603.

Mell, P., Hu, V. and Lippmann, R., 2003. "An Overview of Issues in Testing Intrusion Detection Systems", No. NIST IR 7007, National Institute of Standards and Technology, August 2003.

Rhodes, B. C., Mahaffey, J. A. and Cannady, J. D. 2000. "Multiple Self-Organizing Maps for Intrusion Detection", NIST National Information Systems Security Conference.

Ryan, J., Lin ,M. J. and Miikulainen, R. 1998. "Intrusion Detection with Neural Networks", Advances in Neural Information Processing Systems, Vol 10, pp. 943-949.

WEB_1, 2005. CERT's web site, 17/10/2005. http://www.cert.org.

WEB_2, 2005. InformIT's web site, 15/10/2005. http://www.informit.com.

WEB_3, 2004. Snort's web site, 12/02/2004. http://www.snort.org.

WEB_4 2005. CVE's web site 1/07/2005. http://www.cve.mitre.org.

Zhang, Z., Li J., Manikopoulos, C. N., Jorgenson, J. and Ucles J. 2001. "HIDE: A Hierarchical Network Intrusion Detection System Using Statistical Preprocessing and Neural Network Classification", Proceedings of IEEE Workshop on Information Assurance and Security, West Point, pp. 85-90.

APPENDIX A

LIST OF THE SELECTED VULNERABILITIES

Vulnerabilities selected among top 10 vulnerabilities declared by SANS Institute are given below. Additionally description of the vulnerability, which is taken from Internet site (WEB_4 2005) of common Vulnerabilities and Exposures (CVE) , tool used to exploit the vulnerability and the Snort signature identity of the generated attack are also given below. Since more than one attack may be generated to exploit vulnerability, there may be two or more Snort signature identities associated with each attack generated.

CVE Name	CVE-1999-0002
Description	Buffer overflow in NFS mountd gives root access to remote attackers, mostly in Linux systems.
Snort Signature ID	579
Tool used	Nessus

CVE Name	CVE-1999-0003
Description	Execute commands as root via buffer overflow in Tooltalk database server (rpc.ttdbserverd).
Snort Signature ID	558
Tool used	Nessus

CVE Name	CVE-1999-0021
Description	Arbitrary command execution via buffer overflow in Count.cgi (wwwcount) cgi-bin program.
Snort Signature ID	1149
Tool used	Nessus

CVE Name	CVE-1999-0066
Description	The CGI 'AnyForm2' is installed. Old versions of this CGI have a well known security flaw that lets anyone execute arbitrary commands with the privileges of the http daemon.
Snort Signature ID	892
Tool used	Nessus

CVE Name	CVE-1999-0067
Description	CGI phf program allows remote command execution through shell metacharacters.
Snort Signature ID	1762
Tool used	Nessus

CVE Name	CVE-1999-0070
Description	test-cgi program allows an attacker to list files on the server.
Snort Signature ID	835
Tool used	Nessus

CVE Name	CVE-1999-0146
Description	The campas CGI program provided with some NCSA web servers allows an attacker to read arbitrary files.
Snort Signature ID	847
Tool used	Nessus

CVE Name	CVE-1999-0172
Description	FormMail CGI program allows remote execution of commands.
Snort Signature ID	884
Tool used	Nessus

CVE Name	CVE-1999-0174
Description	The view-source CGI program allows remote attackers to read arbitrary files via a .. (dot dot) attack.
Snort Signature ID	849, 1482
Tool used	Nessus

CVE Name	CVE-1999-0191
Description	The CGI /scripts/tools/newdsn.exe is present. This CGI allows any attacker to create files anywhere on your system if your NTFS permissions are not tight enough, and can be used to overwrite DSNs of existing databases.
Snort Signature ID	1024
Tool used	Nessus

CVE Name	CVE-1999-0208
Description	rpc.ypupdated (NIS) allows remote users to execute arbitrary commands.
Snort Signature ID	661
Tool used	Nessus

CVE Name	CVE-1999-0237
Description	guestbook.pl and guestbook.cgi have a well known security flaw that lets anyone execute arbitrary commands with the privileges of the http daemon.
Snort Signature ID	1597, 1140
Tool used	Nessus

CVE Name	CVE-1999-0260
Description	The jj CGI program allows command execution via shell metacharacters.

Snort Signature ID	1174
Tool used	Nessus

CVE Name	CVE-1999-0262
Description	faxsurvey CGI script on Linux allows remote command execution via shell metacharacters.
Snort Signature ID	857
Tool used	Nessus

CVE Name	CVE-1999-0264
Description	The 'htmlscript' cgi is installed. This CGI has well known security flaw that lets anyone read arbitrary files with the privileges of the http daemon.
Snort Signature ID	826
Tool used	Nessus

CVE Name	CVE-1999-0266
Description	The info2www CGI script allows remote file access or remote command execution.
Snort Signature ID	827
Tool used	Nessus

CVE Name	CVE-1999-0278
Description	It is possible to get the source code of the remote ASP scripts by appending ::$DATA at the end of the request (like GET /default.asp::$DATA).
Snort Signature ID	975
Tool used	Nessus

CVE Name	CAN-1999-0509
Description	Perl, sh, csh, or other shell interpreters are installed in the cgi-bin directory on a WWW site, which allows remote attackers to execute arbitrary commands.
Snort Signature ID	832, 862, 865, 868, 872, 877, 885, 1309, 1648, 2649
Tool used	Nessus, Nikto

CVE Name	CAN-1999-0517
Description	An SNMP community name is the default (e.g. public), null, or missing.
Snort Signature ID	1411, 1893
Tool used	Nessus

CVE Name	CAN-1999-0736
Description	The showcode.asp sample file in IIS and Site Server allows remote attackers to read arbitrary files.
Snort Signature ID	1037
Tool used	Nessus

CVE Name	CVE-1999-1011

Description	The Remote Data Service (RDS) DataFactory component of Microsoft Data Access Components (MDAC) in IIS 3.x and 4.x exposes unsafe methods, which allows remote attackers to execute arbitrary commands.
Snort Signature ID	1023
Tool used	Nessus

CVE Name	CAN-1999-1376
Description	There might be a buffer overflow in the remote fpcount.exe cgi. An attacker may use it to execute arbitrary code on this host.
Snort Signature ID	1012, 1013
Tool used	Nessus

CVE Name	CVE-2000-0208
Description	The htdig (ht://Dig) CGI program htsearch allows remote attackers to read arbitrary files by enclosing the file name with backticks (`) in parameters to htsearch.
Snort Signature ID	1600, 1601, 1602
Tool used	Nessus

CVE Name	CVE-2000-0287
Description	The BizDB CGI script bizdb-search.cgi allows remote attackers to execute arbitrary commands via shell metacharacters in the dbname parameter.
Snort Signature ID	1535
Tool used	Nessus

CVE Name	CVE-2000-0778
Description	This host is running the Microsoft IIS web server. This web server contains a configuration flaw that allows the retrieval of the global.asa file. This file may contain sensitive information such as database passwords, internal addresses, and web application configuration options. This vulnerability may be caused by a missing ISAPI map of the .asa extension to asp.dll.
Snort Signature ID	1016
Tool used	Nessus

CVE Name	CAN-2000-0832
Description	Htgrep CGI program allows remote attackers to read arbitrary files by specifying the full pathname in the hdr parameter.
Snort Signature ID	1207, 1615
Tool used	Nessus

CVE Name	CAN-2000-1081
Description	The xp_displayparamstmt function in SQL Server and Microsoft SQL Server Desktop Engine (MSDE) does not properly restrict the length of a buffer before calling the

	srv_paraminfo function in the SQL Server API for Extended Stored Procedures (XP), which allows an attacker to cause a denial of service or execute arbitrary commands, aka the "Extended Stored Procedure Parameter Parsing" vulnerability.
Snort Signature ID	974
Tool used	xp_displayparamstmt.exe

CVE Name	CAN-2000-1082
Description	The xp_enumresultset function in SQL Server and Microsoft SQL Server Desktop Engine (MSDE) does not properly restrict the length of a buffer before calling the srv_paraminfo function in the SQL Server API for Extended Stored Procedures (XP), which allows an attacker to cause a denial of service or execute arbitrary commands, aka the "Extended Stored Procedure Parameter Parsing" vulnerability.
Snort Signature ID	780
Tool used	xp_enumresultset .exe

CVE Name	CAN-2000-1083
Description	The xp_showcolv function in SQL Server and Microsoft SQL Server Desktop Engine (MSDE) does not properly restrict the length of a buffer before calling the srv_paraminfo function in the SQL Server API for Extended Stored Procedures (XP), which allows an attacker to cause a denial of service or execute arbitrary commands, aka the "Extended Stored Procedure Parameter Parsing" vulnerability.
Snort Signature ID	705
Tool used	xp_showcolv.exe

CVE Name	CAN-2000-1084
Description	The xp_updatecolvbm function in SQL Server and Microsoft SQL Server Desktop Engine (MSDE) does not properly restrict the length of a buffer before calling the srv_paraminfo function in the SQL Server API for Extended Stored Procedures (XP), which allows an attacker to cause a denial of service or execute arbitrary commands, aka the "Extended Stored Procedure Parameter Parsing" vulnerability.
Snort Signature ID	701
Tool used	xp_updatecolvbm.exe

CVE Name	CAN-2000-1085
Description	The xp_peekqueue function in Microsoft SQL Server 2000 and SQL Server Desktop Engine (MSDE) does not properly restrict the length of a buffer before calling the srv_paraminfo function in the SQL Server API for Extended Stored Procedures (XP), which allows an attacker to cause a denial of service or execute arbitrary commands, aka the "Extended

	Stored Procedure Parameter Parsing" vulnerability.
Snort Signature ID	760
Tool used	xp_peekqueue.sql / xp_peekqueue .exe

CVE Name	CAN-2000-1086
Description	The xp_printstatements function in Microsoft SQL Server 2000 and SQL Server Desktop Engine (MSDE) does not properly restrict the length of a buffer before calling the srv_paraminfo function in the SQL Server API for Extended Stored Procedures (XP), which allows an attacker to cause a denial of service or execute arbitrary commands, aka the "Extended Stored Procedure Parameter Parsing" vulnerability.
Snort Signature ID	699
Tool used	xp_printstatements .exe

CVE Name	CAN-2000-1087
Description	The xp_proxiedmetadata function in Microsoft SQL Server 2000 and SQL Server Desktop Engine (MSDE) does not properly restrict the length of a buffer before calling the srv_paraminfo function in the SQL Server API for Extended Stored Procedures (XP), which allows an attacker to cause a denial of service or execute arbitrary commands, aka the "Extended Stored Procedure Parameter Parsing" vulnerability.
Snort Signature ID	707
Tool used	xp_proxiedmetadata.exe

CVE Name	CAN-2000-1209
Description	The "sa" account is installed with a default null password on (1) Microsoft SQL Server 2000, (2) SQL Server 7.0, and (3) Data Engine (MSDE) 1.0, including third party packages that use these products such as (4) Tumbleweed Secure Mail (MMS) (5) Compaq Insight Manager, and (6) Visio 2000, which allows remote attackers to gain privileges, as exploited by worms such as Voyager Alpha Force and Spida.
Snort Signature ID	688
Tool used	Nessus

CVE Name	CVE-2001-0236
Description	Buffer overflow in Solaris snmpXdmid SNMP to DMI mapper daemon allows remote attackers to execute arbitrary commands via a long "indication" event.
Snort Signature ID	1279
Tool used	Nessus

CVE Name	CVE-2001-0241
Description	Buffer overflow in Internet Printing ISAPI extension in Windows 2000 allows remote attackers to gain root privileges

	via a long print request that is passed to the extension through IIS 5.0.
Snort Signature ID	971
Tool used	CVE-2001-0241.pl

CVE Name	CVE-2001-0333
Description	When IIS receives a user request to run a script, it renders the request in a decoded canonical form, then performs security checks on the decoded request. A vulnerability results because a second, superfluous decoding pass is performed after the initial security checks are completed. Thus, a specially crafted request could allow an attacker to execute arbitrary commands on the IIS Server.
Snort Signature ID	970
Tool used	Nessus

CVE Name	CVE-2001-0717
Description	Format string vulnerability in ToolTalk database server rpc.ttdbserverd allows remote attackers to execute arbitrary commands via format string specifiers that are passed to the syslog function.
Snort Signature ID	588
Tool used	Nessus

CVE Name	CVE-2001-0779
Description	Buffer overflow in rpc.yppasswdd (yppasswd server) in Solaris 2.6, 7 and 8 allows remote attackers to gain root access via a long username.
Snort Signature ID	2025
Tool used	Nessus

CVE Name	CAN-2002-0012
Description	Vulnerabilities in a large number of SNMP implementations allow remote attackers to cause a denial of service or gain privileges via SNMPv1 trap handling, as demonstrated by the PROTOS c06-SNMPv1 test suite. NOTE: It is highly likely that this candidate will be SPLIT into multiple candidates, one or more for each vendor. This and other SNMP-related candidates will be updated when more accurate information is available.
Snort Signature ID	1412, 1413, 1417, 1418, 1419, 1420, 1421
Tool used	Nessus

CVE Name	CVE-2002-0033
Description	Heap-based buffer overflow in cfsd_calloc function of Solaris cachefsd allows remote attackers to execute arbitrary code via a request with a long directory and cache name.
Snort Signature ID	1746
Tool used	Nessus

CVE Name	CAN-2002-0071
Description	Buffer overflow in the ism.dll ISAPI extension that implements HTR scripting in Internet Information Server (IIS) 4.0 and 5.0 allows attackers to cause a denial of service or execute arbitrary code via HTR requests with long variable names.
Snort Signature ID	1618, 1807
Tool used	Nessus

CVE Name	CAN-2002-0073
Description	The FTP service in Internet Information Server (IIS) 4.0, 5.0 and 5.1 allows attackers who have established an FTP session to cause a denial of service via a specially crafted status request containing glob characters.
Snort Signature ID	1777
Tool used	Nessus

CVE Name	CAN-2002-0079
Description	Buffer overflow in the chunked encoding transfer mechanism in Internet Information Server (IIS) 4.0 and 5.0 Active Server Pages allows attackers to cause a denial of service or execute arbitrary code.
Snort Signature ID	1618, 1807
Tool used	Nessus

CVE Name	CAN-2002-0364
Description	Buffer overflow in the chunked encoding transfer mechanism in IIS 4.0 and 5.0 allows attackers to execute arbitrary code via the processing of HTR request sessions, aka "Heap Overrun in HTR Chunked Encoding Could Enable Web Server Compromise".
Snort Signature ID	1806
Tool used	Nessus

CVE Name	CVE-2002-0392
Description	Apache 1.3 through 1.3.24, and Apache 2.0 through 2.0.36, allows remote attackers to cause a denial of service and possibly execute arbitrary code via a chunk-encoded HTTP request that causes Apache to use an incorrect size.
Snort Signature ID	1807
Tool used	Nessus

CVE Name	CAN-2002-0421
Description	IIS 4.0 allows local users to bypass the "User cannot change password" policy for Windows NT by directly calling .htr password changing programs in the /iisadmpwd directory, including (1) aexp2.htr, (2) aexp2b.htr, (3) aexp3.htr , or (4)

	aexp4.htr.
Snort Signature ID	1487
Tool used	Nessus

CVE Name	CAN-2002-0649
Description	Multiple buffer overflows in SQL Server 2000 Resolution Service allow remote attackers to cause a denial of service or execute arbitrary code via UDP packets to port 1434 in which (1) a 0x04 byte causes the SQL Monitor thread to generate a long registry key name, or (2) a 0x08 byte with a long string causes heap corruption.
Snort Signature ID	2050
Tool used	Nessus

CVE Name	CAN-2002-0682
Description	Cross-site scripting vulnerability in Apache Tomcat 4.0.3 allows remote attackers to execute script as other web users via script in a URL with the /servlet/ mapping, which does not filter the script when an exception is thrown by the servlet.
Snort Signature ID	1827
Tool used	Nessus

CVE Name	CAN-2002-1142
Description	Heap-based buffer overflow in the Remote Data Services (RDS) component of Microsoft Data Access Components (MDAC) 2.1 through 2.6, and Internet Explorer 5.01 through 6.0, allows remote attackers to execute code via a malformed HTTP request to the Data Stub.
Snort Signature ID	1970
Tool used	Nessus

CVE Name	CAN-2002-1232
Description	Memory leak in ypdb_open in yp_db.c for ypserv before 2.5 in the NIS package 3.9 and earlier allows remote attackers to cause a denial of service (memory consumption) via a large number of requests for a map that does not exist.
Snort Signature ID	590
Tool used	Nessus

CVE Name	CAN-2003-0028
Description	Integer overflow in the xdrmem_getbytes() function, and possibly other functions, of XDR (external data representation) libraries derived from SunRPC, including libnsl, libc, glibc, and dietlibc, allows remote attackers to execute arbitrary code via certain integer values in length fields, a different vulnerability than CAN-2002-0391.
Snort Signature ID	2092
Tool used	Nessus

CVE Name	CAN-2003-0109
Description	Buffer overflow in ntdll.dll, as used by WebDAV on Windows 2000, allows remote attackers to execute arbitrary code.via a long request to IIS 5.0.
Snort Signature ID	2091
Tool used	Nessus

CVE Name	CAN-2003-0118
Description	SQL injection vulnerability in the Document Tracking and Administration (DTA) website of Microsoft BizTalk Server 2000 and 2002 allows remote attackers to execute operating system commands via a request to (1) rawdocdata.asp or (2) RawCustomSearchField.asp containing an embedded SQL statement.
Snort Signature ID	2133
Tool used	Nessus

CVE Name	CAN-2003-0227 / CAN-2003-0349
Description	Some versions of IIS shipped with a default file, nsiislog.dll, within the /scripts directory. Nessus has determined that the remote host has the file installed. The NSIISLOG.dll CGI may allow an attacker to execute arbitrary commands on this host, through a buffer overflow.
Snort Signature ID	2129
Tool used	Nessus

CVE Name	CAN-2003-0377
Description	SQL injection vulnerability in the web-based administration interface for iisPROTECT 2.2-r4, and possibly earlier versions, allows remote attackers to insert arbitrary SQL and execute code via certain variables, as demonstrated using the GroupName variable in SiteAdmin.ASP.
Snort Signature ID	2130
Tool used	Nikto, Shadow Security Scanner

CVE Name	CAN-2003-0605
Description	The RPC DCOM interface in Windows 2000 SP3 and SP4 allows remote attackers to cause a denial of service (crash), and local attackers to use the DoS to hijack the epmapper pipe to gain privileges, via certain messages to the __RemoteGetClassObject interface that cause a NULL pointer to be passed to the PerformScmStage function.
Snort Signature ID	2251
Tool used	Nessus

CVE Name	CAN-2003-0818
Description	Multiple integer overflows in Microsoft ASN.1 library (MSASN1.DLL), as used in LSASS.EXE, CRYPT32.DLL,

	and other Microsoft executables and libraries on Windows NT 4.0, 2000, and XP, allow remote attackers to execute arbitrary code via ASN.1 BER encodings with very large length fields that cause arbitrary heap data to be overwritten.
Snort Signature ID	2383, 2386
Tool used	Nessus

APPENDIX B

LIST OF THE EXPERIMENTAL RESULTS

Tables of raw data obtained after the test step is given this section. For the results of the 3-layer MLP networks, first column of the table shows the architecture of the neural network, while first row of the table represents the identity of the neural network, as 10 neural networks were generated for each of the neural network architecture. Reason for generating 10 neural networks for each of the neural network architecture is to avoid poor performances due to the bad initialization of the neural network weights.

For the results of the 4-layer MLP networks, first column of the table represents the number of nodes in input, first and second hidden layers of the neural network architecture, while first row of the table represents number of nodes in the third hidden layer and in the output layer. Intersection of a row header and a column header represents the number of false positives of a neural network whose architecture can be obtained by joining row header and column header. For instance, intersection of the row header of 7-5-5 and column header of 5-1 represents the false positives of the neural network whose architecture is 7-5-5-5-1. As managing whole neural networks would be a troublesome task and training periods would take long, 4-layer backpropagation networks were not re-initialized.

Table B.1. Number of false positives for 3-layer backpropagation networks.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
7-5-5-1	1	124	140	0	0	1	0	2	0	0
7-5-10-1	9	20	60	4	0	110	17	0	46	144
7-5-15-1	1	0	18	0	1	53	9	0	15	1
7-5-20-1	0	0	0	0	67	6	2	6	187	0
7-5-25-1	0	4	139	0	31	0	1	62	0	1
7-10-5-1	0	64	0	133	0	12	35	13	72	0
7-10-10-1	14	82	12	0	0	19	1	24	13	1
7-10-15-1	64	1	1	0	0	0	15	60	0	1
7-10-20-1	1	9	99	0	19	14	0	4	1	0
7-10-25-1	0	1	0	1	16	6	23	14	1	1

(cont. on next page)

Table B.1 (cont.)

7-15-5-1	27	0	16	65	1	0	0	0	0	39
7-15-10-1	0	0	10	62	7	24	0	61	0	0
7-15-15-1	0	1	0	1	2	1	94	1	1	0
7-15-20-1	0	0	61	6	13	76	11	0	1	16
7-15-25-1	159	2	0	1	0	93	1	200	0	1
7-20-5-1	0	1	1	0	104	76	40	2	0	0
7-20-10-1	0	15	1	18	7	0	9	1	1	1
7-20-15-1	7	57	21	23	1	0	1	0	27	9
7-20-20-1	23	15	14	1	0	2	16	1	0	0
7-20-25-1	0	1	1	1	0	3	18	2	3	1
7-25-5-1	8	0	45	1	0	1	1	1	1	61
7-25-10-1	0	0	62	1	0	0	73	2	1	0
7-25-15-1	8	0	1	61	1	56	74	1	3	0
7-25-20-1	51	0	16	1	11	3	10	1	0	82
7-25-25-1	1	2	1	1	14	10	0	1	36	18

Table B.2. Number of false negatives for 3-layer backpropagation networks.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
7-5-5-1	967	951	970	971	971	329	971	967	971	560
7-5-10-1	329	329	551	557	350	656	504	971	656	644
7-5-15-1	330	971	971	552	329	646	868	971	642	330
7-5-20-1	645	971	550	971	531	971	332	971	650	971
7-5-25-1	348	223	261	656	971	330	329	245	652	656
7-10-5-1	656	330	338	551	656	329	971	551	339	656
7-10-10-1	638	326	647	645	329	970	327	971	331	329
7-10-15-1	437	329	329	553	329	330	971	338	536	325
7-10-20-1	327	227	329	350	329	239	329	261	329	971
7-10-25-1	226	330	656	329	656	329	325	329	329	329
7-15-5-1	971	656	326	326	356	326	329	335	329	596
7-15-10-1	644	325	329	323	329	329	329	140	329	330
7-15-15-1	560	329	329	330	242	329	333	329	329	329
7-15-20-1	655	329	330	329	329	224	226	331	326	328
7-15-25-1	248	329	326	254	329	176	314	215	326	246
7-20-5-1	224	329	325	656	534	222	328	551	971	325
7-20-10-1	329	226	329	329	325	329	329	329	329	329
7-20-15-1	326	316	226	325	329	329	227	253	329	245
7-20-20-1	325	226	258	329	326	327	329	326	329	329
7-20-25-1	323	233	325	329	326	329	234	326	326	325
7-25-5-1	178	239	329	329	330	325	325	301	350	551
7-25-10-1	227	226	224	325	553	325	329	329	329	247
7-25-15-1	226	329	326	237	330	213	226	226	326	326
7-25-20-1	323	329	325	226	194	326	277	329	326	279
7-25-25-1	327	329	325	329	236	329	329	329	226	186

Table B.3. Number of false positives for 3-layer Levenberg-Marquardt networks.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
7-5-5-1	5	13	18	9	11	10	10	8	0	107
7-5-10-1	1	2	9450	17	0	17	69	28	70	65
7-5-15-1	19	9450	9	23	17	7	52	29	10	9450
7-5-20-1	36	9450	26	8	3	18	3	13	5	14
7-5-25-1	4	8	4	20	37	0	30	9	17	14
7-10-5-1	6	1	1	0	9	0	3	14	2	8
7-10-10-1	1	16	15	1	0	9	8	17	4	9450
7-10-15-1	1	3	4	9	4	0	17	0	2	0
7-10-20-1	5	2	3	14	12	2	17	13	3	5
7-10-25-1	6	1	7	0	18	0	20	13	11	6
7-15-5-1	0	6	13	7	3	0	1	4	0	0
7-15-10-1	6	2498	0	2	0	6	3	0	7	0
7-15-15-1	0	0	2	9	0	3	0	4	5	0
7-15-20-1	4	0	1	0	0	3	0	0	2	1
7-15-25-1	3	3	0	4	0	3	2	0	1	2
7-20-5-1	3	1	0	0	1	11	9	0	0	1
7-20-10-1	3	0	6	0	9450	0	0	0	0	2
7-20-15-1	1	0	0	0	10	1	0	0	9	0
7-20-20-1	0	1	1	0	0	0	0	0	1	1
7-20-25-1	3	0	0	0	1	161	2	1	0	1
7-25-5-1	0	0	0	0	1	1	0	0	0	0
7-25-10-1	0	0	0	0	0	2	0	0	0	3
7-25-15-1	1	0	2	1	0	0	8	0	1	0
7-25-20-1	0	1	2	0	0	0	0	0	0	0
7-25-25-1	3	0	1	0	1	0	3	1	0	0

Table B.4. Number of false negatives for 3-layer Levenberg-Marquardt networks.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
7-5-5-1	23	16	1	17	13	186	22	13	7	247
7-5-10-1	11	1	0	88	866	7	175	49	22	228
7-5-15-1	74	0	16	4	13	6	114	24	15	0
7-5-20-1	99	0	16	18	11	92	154	75	2	99
7-5-25-1	13	10	7	13	87	13	22	10	7	14
7-10-5-1	23	20	7	0	8	17	20	323	21	13
7-10-10-1	14	656	1	1	0	0	7	7	13	0
7-10-15-1	2	1	2	15	1	7	1	1	1	0
7-10-20-1	1	1	7	8	6	6	13	4	1	15
7-10-25-1	8	1	0	0	14	0	6	7	1	8
7-15-5-1	800	1	16	2	188	13	2	19	0	13
7-15-10-1	1	101	0	0	0	134	1	0	8	6
7-15-15-1	0	0	6	3	0	0	0	2	1	0
7-15-20-1	7	0	1	0	0	1	1	0	7	0

(cont. on next page)

Table B.4 (cont.)

7-15-25-1	1	0	0	0	0	1	1	6	0	0
7-20-5-1	6	0	0	6	9	7	7	13	0	0
7-20-10-1	0	0	7	0	0	0	1	0	0	0
7-20-15-1	0	1	0	0	1	0	0	0	326	1
7-20-20-1	0	1	1	0	0	0	0	0	7	0
7-20-25-1	7	0	1	0	0	66	1	0	0	0
7-25-5-1	6	7	1	0	0	6	7	0	7	7
7-25-10-1	1	1	0	0	0	12	0	0	0	7
7-25-15-1	0	0	0	7	0	0	7	0	0	0
7-25-20-1	0	0	0	0	0	0	0	0	0	0
7-25-25-1	0	0	1	1	1	0	3	0	0	0

Table B.5. Number of false positives for 4-layer backpropagation networks.

	-5-1	**-10-1**	**-15-1**	**-20-1**	**-25-1**
7-5-5	0	0	61	0	133
7-5-10	67	1	0	0	0
7-5-15	7	12	4	46	0
7-5-20	0	135	14	77	0
7-5-25	0	1	106	1	25
7-10-5	1	1	116	1	4
7-10-10	0	32	18	5	78
7-10-15	5	27	7	62	0
7-10-20	69	1	0	1	1
7-10-25	64	41	0	1	1
7-15-5	1	1	1	0	65
7-15-10	1	103	64	3	0
7-15-15	1	18	17	62	1
7-15-20	95	1	62	0	0
7-15-25	1	15	0	0	18
7-20-5	7	13	11	32	2
7-20-10	2	92	0	10	49
7-20-15	0	14	1	0	12
7-20-20	11	12	1	0	1
7-20-25	28	16	19	0	1
7-25-5	116	0	16	1	47
7-25-10	1	0	0	0	1
7-25-15	0	15	17	55	30
7-25-20	0	1	10	1	63
7-25-25	17	2	0	1	11

Table B.6. Number of false negatives for 4-layer backpropagation networks.

	-5-1	**-10-1**	**-15-1**	**-20-1**	**-25-1**
7-5-5	971	971	551	329	655
7-5-10	329	656	326	564	971
7-5-15	971	655	328	963	935
7-5-20	656	218	322	326	644
7-5-25	971	226	330	224	307
7-10-5	329	644	219	213	320
7-10-10	329	329	213	652	224
7-10-15	330	312	329	224	325
7-10-20	557	553	329	242	329
7-10-25	224	649	652	328	325
7-15-5	656	329	311	226	224
7-15-10	336	329	312	329	323
7-15-15	329	343	277	323	224
7-15-20	226	224	245	226	245
7-15-25	329	325	327	226	325
7-20-5	331	199	329	215	550
7-20-10	656	329	318	329	304
7-20-15	329	322	276	330	242
7-20-20	326	160	329	224	242
7-20-25	226	325	329	329	656
7-25-5	329	330	326	226	86
7-25-10	329	329	329	326	224
7-25-15	635	224	193	325	327
7-25-20	326	309	326	276	137
7-25-25	198	321	329	245	95

Table B.7. Number of false positives for 4-Layer Levenberg-Marquardt networks.

	-5-1	**-10-1**	**-15-1**	**-20-1**	**-25-1**
7-5-5	9450	24	11	3	2
7-5-10	18	32	20	19	1
7-5-15	0	82	4	9	1
7-5-20	0	11	1	3	14
7-5-25	0	7	1	0	9
7-10-5	0	0	0	4	1
7-10-10	3	0	1	0	0
7-10-15	0	0	3	0	0
7-10-20	0	0	0	0	1
7-10-25	0	9450	0	0	10
7-15-5	1	0	222	0	19
7-15-10	0	0	11	0	0
7-15-15	0	3	0	0	1
7-15-20	0	0	0	0	3

(cont. on next page)

Table B.7 (cont.)

7-15-25	0	0	0	1	0
7-20-5	0	0	0	9	0
7-20-10	0	0	0	0	0
7-20-15	0	0	0	0	0
7-20-20	0	0	3	0	3
7-20-25	0	0	0	0	0
7-25-5	14	0	1	0	0
7-25-10	0	0	3	0	0
7-25-15	3	0	0	0	0
7-25-20	0	0	9450	0	0
7-25-25	1	0	0	0	0

Table B.8. Number of false negatives for 4-layer Levenberg-Marquardt networks.

	-5-1	**-10-1**	**-15-1**	**-20-1**	**-25-1**
7-5-5	0	94	16	18	22
7-5-10	3	191	7	27	6
7-5-15	8	45	9	1	6
7-5-20	1	14	7	0	162
7-5-25	0	1	0	1	6
7-10-5	1	6	7	157	7
7-10-10	2	6	0	0	0
7-10-15	6	1	0	0	0
7-10-20	6	0	0	1	0
7-10-25	13	0	6	1	1
7-15-5	329	0	81	6	270
7-15-10	1	0	0	0	0
7-15-15	0	1	0	0	0
7-15-20	6	0	1	0	11
7-15-25	0	0	0	13	1
7-20-5	1	0	0	1	1
7-20-10	0	0	0	1	0
7-20-15	1	0	0	1	1
7-20-20	0	0	71	0	6
7-20-25	0	6	0	0	1
7-25-5	870	1	0	1	0
7-25-10	0	0	6	0	0
7-25-15	1	0	0	0	0
7-25-20	0	1	0	0	0
7-25-25	13	0	0	0	0

Table B.9. Number of false positives for 3-layer backpropagation networks.

	1	2	3	4	5	6	7	8	9	10
7-5-5-1	4233	4233	0	0	0	4233	4233	0	4233	0
7-5-10-1	0	0	4233	0	4233	4233	0	0	4233	0
7-5-15-1	4233	0	0	0	0	0	0	4233	0	4233
7-5-20-1	0	0	0	0	4233	0	0	4233	0	0
7-5-25-1	4233	4233	0	4233	0	4233	4233	0	4233	4233
7-10-5-1	0	0	0	0	0	0	0	4233	0	4233
7-10-10-1	0	0	0	0	0	4233	0	0	0	0
7-10-15-1	4233	0	4233	4233	0	0	0	4233	0	0
7-10-20-1	0	0	0	4233	0	4233	0	4233	0	0
7-10-25-1	0	0	0	0	0	0	0	4233	0	4233
7-15-5-1	0	0	0	0	0	0	0	0	4233	0
7-15-10-1	0	4233	0	0	4233	0	0	0	4233	0
7-15-15-1	0	0	0	0	4233	0	4233	4233	0	0
7-15-20-1	0	0	0	0	4233	0	0	0	4233	0
7-15-25-1	0	0	0	0	0	0	4233	4233	0	0
7-20-5-1	0	0	4233	0	4233	0	0	4233	0	4233
7-20-10-1	0	0	0	0	0	0	0	0	0	0
7-20-15-1	0	0	0	0	0	4233	0	0	0	0
7-20-20-1	4233	0	0	0	0	4233	0	0	0	0
7-20-25-1	0	4233	0	0	0	0	4233	4233	0	0
7-25-5-1	0	4233	0	0	4233	0	0	4233	4233	0
7-25-10-1	0	0	4233	0	4233	0	4233	0	0	4233
7-25-15-1	0	0	0	0	0	0	4233	0	0	0
7-25-20-1	4233	0	0	0	0	4233	0	0	4233	0
7-25-25-1	0	4233	0	0	4233	0	4233	0	0	0

Table B.10. Number of false negatives for 3-layer backpropagation networks.

	1	2	3	4	5	6	7	8	9	10
7-5-5-1	73	46	16	73	73	63	63	73	73	57
7-5-10-1	0	36	73	30	73	33	0	73	16	6
7-5-15-1	63	73	73	63	63	63	73	73	36	63
7-5-20-1	63	73	73	73	63	73	43	73	73	73
7-5-25-1	73	36	63	73	73	73	63	63	73	46
7-10-5-1	73	73	63	33	73	63	73	73	63	73
7-10-10-1	63	63	73	63	63	73	63	73	73	30
7-10-15-1	73	63	63	73	63	63	73	63	73	36
7-10-20-1	30	63	63	73	63	57	63	67	73	73
7-10-25-1	57	36	73	63	73	36	57	63	63	63
7-15-5-1	73	73	63	36	63	63	73	73	63	63
7-15-10-1	57	63	63	36	63	63	63	57	73	67
7-15-15-1	63	30	27	36	0	65	73	63	63	63
7-15-20-1	73	27	36	36	73	63	63	43	63	0

(cont. on next page)

Table B.10 (cont.)

7-15-25-1	27	30	63	57	63	63	63	57	36	0
7-20-5-1	63	63	36	73	73	30	0	73	73	63
7-20-10-1	36	36	36	63	36	63	46	57	65	63
7-20-15-1	63	59	57	57	36	73	63	6	71	30
7-20-20-1	8	63	36	63	63	63	63	57	63	63
7-20-25-1	57	36	63	63	63	63	63	63	63	0
7-25-5-1	73	33	67	63	73	36	63	7	63	73
7-25-10-1	57	63	36	36	73	36	36	63	63	73
7-25-15-1	36	63	63	73	63	36	36	30	30	63
7-25-20-1	63	63	63	73	36	31	0	63	57	63
7-25-25-1	30	63	36	63	30	65	36	63	73	57

Table B.11. Number of false positives for 3-layer Levenberg-Marquardt networks.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
7-5-5-1	4233	0	4233	0	0	0	0	0	4233	4233
7-5-10-1	4233	0	4233	0	4233	4233	4233	0	0	4233
7-5-15-1	4233	4233	4233	4233	0	4233	0	0	0	0
7-5-20-1	4233	4233	0	0	0	0	4233	4233	0	0
7-5-25-1	0	4233	4233	0	4233	0	0	4233	0	0
7-10-5-1	4233	4233	0	4233	4233	0	4233	4233	0	4233
7-10-10-1	4233	4233	0	4233	0	4233	4233	0	0	4233
7-10-15-1	0	4233	4233	4233	0	4233	0	0	0	4233
7-10-20-1	4233	4233	4233	4233	0	4233	0	0	4233	4233
7-10-25-1	4233	0	4233	0	0	0	0	4233	4233	0
7-15-5-1	4233	4233	0	0	0	0	0	4233	0	0
7-15-10-1	4233	4233	4233	4233	4233	0	0	4233	4233	0
7-15-15-1	0	4233	4233	4233	0	4233	0	0	0	4233
7-15-20-1	4233	0	0	0	0	0	4233	0	0	0
7-15-25-1	4233	4233	4233	0	0	4233	4233	0	4233	0
7-20-5-1	4233	0	4233	0	4233	4233	4233	4233	4233	0
7-20-10-1	0	4233	0	0	4233	4233	0	4233	0	0
7-20-15-1	0	4233	4233	0	0	4233	0	4233	0	4233
7-20-20-1	0	4233	4233	0	0	0	4233	0	4233	0
7-20-25-1	0	4233	4233	4233	0	4233	0	0	0	4233
7-25-5-1	0	4233	4233	4233	0	0	0	0	4233	4233
7-25-10-1	0	0	4233	4233	4233	0	4233	0	4233	0
7-25-15-1	4233	4233	4233	4233	4233	0	0	4233	4233	4233
7-25-20-1	4233	0	0	0	4233	4233	4233	4233	4233	4233
7-25-25-1	4233	4233	4233	0	0	0	0	4233	4233	4233

Table B.12. Number of false negatives for 3-layer Levenberg-Marquardt networks.

	1	2	3	4	5	6	7	8	9	10
7-5-5-1	10	57	30	36	63	30	63	65	30	67
7-5-10-1	57	30	0	65	73	63	73	73	63	36
7-5-15-1	0	0	67	57	30	30	36	73	0	0
7-5-20-1	36	0	36	30	24	36	73	4	57	73
7-5-25-1	30	1	59	46	27	30	36	63	57	30
7-10-5-1	73	57	67	37	57	36	57	30	30	36
7-10-10-1	8	73	57	73	57	0	30	36	50	0
7-10-15-1	36	30	0	30	30	30	27	63	57	10
7-10-20-1	0	0	30	0	0	6	0	63	0	0
7-10-25-1	27	36	0	30	30	0	30	63	30	36
7-15-5-1	73	30	30	0	63	67	30	30	73	30
7-15-10-1	30	73	10	27	57	30	36	46	30	57
7-15-15-1	57	18	0	27	30	63	27	30	38	57
7-15-20-1	57	57	27	73	27	18	65	63	30	30
7-15-25-1	0	30	6	30	0	57	0	0	30	0
7-20-5-1	57	57	57	67	30	63	30	63	36	30
7-20-10-1	30	63	30	30	0	30	63	57	30	57
7-20-15-1	0	30	0	30	36	0	30	57	63	30
7-20-20-1	57	30	27	36	8	30	0	30	29	30
7-20-25-1	6	30	0	27	27	57	27	57	6	30
7-25-5-1	30	36	6	30	57	30	36	33	36	46
7-25-10-1	0	0	30	30	30	57	27	63	63	0
7-25-15-1	63	30	0	27	0	57	30	59	6	0
7-25-20-1	30	36	27	63	0	57	0	57	30	30
7-25-25-1	63	0	30	30	67	73	57	18	57	0

Table B.13. Number of false positives for 4-layer backpropagation networks.

	-5-1	-10-1	-15-1	-20-1	-25-1
7-5-5	0	0	0	0	4233
7-5-10	0	0	0	0	0
7-5-15	0	4233	0	0	0
7-5-20	0	0	0	0	0
7-5-25	0	0	0	0	0
7-10-5	0	0	4233	4233	0
7-10-10	0	0	4233	0	0
7-10-15	0	0	0	4233	0
7-10-20	0	0	0	0	0
7-10-25	0	4233	0	0	0
7-15-5	0	0	0	0	4233
7-15-10	0	0	0	4233	0
7-15-15	0	0	0	4233	4233
7-15-20	0	0	0	4233	0
7-15-25	0	4233	0	0	0

(cont. on next page)

Table B.13 (cont.)

7-20-5	0	0	0	0	0
7-20-10	4233	0	0	0	0
7-20-15	0	0	0	0	0
7-20-20	4233	4233	0	0	0
7-20-25	0	4233	0	0	4233
7-25-5	0	0	0	4233	4233
7-25-10	4233	0	0	0	4233
7-25-15	0	4233	4233	0	4233
7-25-20	0	4233	0	0	0
7-25-25	4233	0	0	0	0

Table B.14. Number of false negatives for 4-layer backpropagation networks.

	-5-1	**-10-1**	**-15-1**	**-20-1**	**-25-1**
7-5-5	73	73	73	63	6
7-5-10	57	73	63	63	73
7-5-15	37	46	63	57	73
7-5-20	73	30	36	36	63
7-5-25	73	36	27	36	57
7-10-5	63	63	57	73	30
7-10-10	30	30	6	73	63
7-10-15	63	30	36	33	63
7-10-20	0	73	6	57	57
7-10-25	63	73	65	36	30
7-15-5	46	36	0	73	63
7-15-10	63	63	63	63	36
7-15-15	63	63	63	36	36
7-15-20	33	63	63	73	57
7-15-25	0	30	63	63	67
7-20-5	63	63	73	63	73
7-20-10	73	57	73	71	36
7-20-15	36	63	0	63	30
7-20-20	54	63	63	73	30
7-20-25	36	63	57	63	73
7-25-5	36	46	57	36	0
7-25-10	6	36	63	63	30
7-25-15	57	63	57	63	36
7-25-20	36	30	36	6	36
7-25-25	6	30	63	63	63

Table B.15. Number of false positives for 4-layer Levenberg-Marquardt networks.

	-5-1	**-10-1**	**-15-1**	**-20-1**	**-25-1**
7-5-5	4233	4233	0	0	4233
7-5-10	0	4233	0	4233	4233
7-5-15	0	0	0	4233	4233
7-5-20	0	4233	0	4233	0
7-5-25	0	4233	0	0	0
7-10-5	0	4233	4233	0	4233
7-10-10	0	4233	4233	4233	0
7-10-15	4233	0	4233	4233	0
7-10-20	0	0	4233	4233	0
7-10-25	4233	4233	4233	0	4233
7-15-5	0	4233	4233	4233	0
7-15-10	4233	4233	0	0	0
7-15-15	0	4233	4233	0	4233
7-15-20	0	4233	0	4233	0
7-15-25	0	4233	4233	4233	0
7-20-5	4233	4233	0	0	4233
7-20-10	0	4233	4233	0	4233
7-20-15	0	0	0	0	0
7-20-20	4233	0	4233	0	4233
7-20-25	4233	4233	4233	0	0
7-25-5	0	0	0	0	4233
7-25-10	4233	4233	0	0	4233
7-25-15	0	0	0	4233	4233
7-25-20	4233	0	4233	4233	4233
7-25-25	4233	4233	4233	0	4233

Table B.16. Number of false negatives for 4-layer Levenberg-Marquardt networks.

	-5-1	**-10-1**	**-15-1**	**-20-1**	**-25-1**
7-5-5	0	36	52	0	36
7-5-10	30	73	30	40	57
7-5-15	63	30	0	57	57
7-5-20	73	30	53	30	36
7-5-25	67	63	40	30	36
7-10-5	1	57	31	36	36
7-10-10	57	57	57	40	0
7-10-15	36	57	0	63	30
7-10-20	30	30	0	57	57
7-10-25	57	0	57	0	57
7-15-5	63	0	30	57	33
7-15-10	57	33	57	63	30
7-15-15	57	57	57	31	30
7-15-20	57	30	57	30	6

(cont. on next page)

Table B.16 (cont.)

7-15-25	27	57	63	63	57
7-20-5	63	46	30	36	36
7-20-10	57	37	30	57	30
7-20-15	30	0	30	27	27
7-20-20	0	33	40	57	30
7-20-25	0	33	59	0	57
7-25-5	73	30	52	0	57
7-25-10	57	57	30	30	30
7-25-15	57	51	27	57	27
7-25-20	57	38	0	63	67
7-25-25	30	30	33	0	27