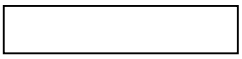
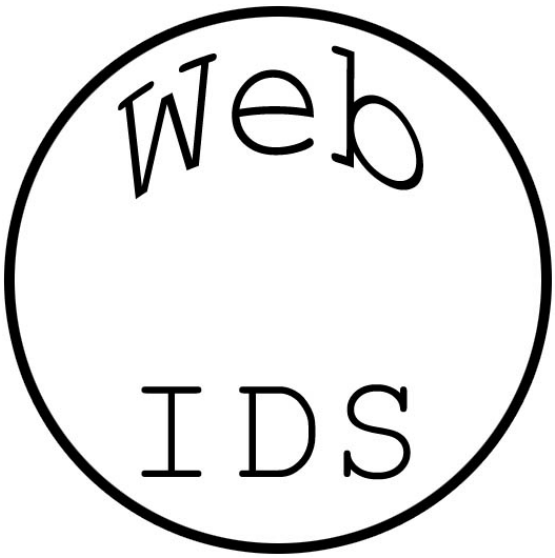
David Ellis

Candidate No: 

COGS Final Year Project
Computer Science and Artificial Intelligence

Supervisor: Ian Wakeman

May 2002



An Intrusion Detection System Model for Web Servers using Artificial Neural Networks

Statement of Originality

This report is submitted as part requirement for the degree of Computer Science and Artificial Intelligence at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.

Signed:

1 Acknowledgements

Thanks to Ian Wakeman for his thoughtful guidance and excellent supervision throughout the project.

Thank you to Gavin Foster for pointers regarding the data mining aspects of the project.

Thanks for the Schmoo team for the data from the DEFCON CCTF competition.

Thanks to the Nessus team for the use of their vulnerability scanner.

Thanks to Geoff Davies from ISEC (at the innovation center) for the pointers he gave. As well as the kind offer of trace data for the project.

Thank you Danielle Daugan and Caroline Auric for their help with the statistics work.

Thank you to Sampsa Sojaka for advice and inspiration throughout the project.

2 Summary

This project investigates the field of Intrusion Detection. In particular it looks at attacks against the Web Server, which arrive over the HTTP protocol. After reading about a number of exploitable programs which typically reside on a Web Server an interesting observation was made, it seems that the URL of an attack is very different from that of a normal request. With this in mind, an appropriate representation of the URL string was formed and a number of Analysis techniques were investigated.

A framework with which to carry out the investigation within was designed and implemented. It provides us with an environment for testing a variety of techniques in order to distinguish between a normal URL and an attack. The framework consists of two main sections, the Event Engine and the Analysis Engine.

The Event Engine provides us with a way to read data into the system and fire appropriate events on the introduction of and analysis of data. Three such readers were implemented, a Packet Sniffer which reads raw data from the network interface or appropriate trace file, a Proxy Server which reads from a TCP/IP socket and a reader which reads from a log file format.

The Analysis Engine provides a way to analyze the data and pass the results of the analysis back to the Event Engine. Several types of Neural Network were incorporated into the system. Including a Multi-Layer Perceptron (MLP) implemented with Back Propagation and a Genetic Algorithm as training algorithms. As well as an unsupervised Neural Network implementing Sangas rule for performing PCA on the data, the K-Nearest Neighbours algorithm was used here to perform the final classification of the data. Other techniques that were tested included the use of simple occurrences of important letter groups; such as vowels and digits. Finally the investigation involved a look at recurrent Neural Networks namely the Elman Network was investigated.

Each of the techniques was tested on a number of Web Sites each of which varied in the complexity of the URL string. We ranged the sites from simple static web sites to highly complex sites with database driven back-ends.

We found very promising results came from the use of MLP networks. The Neural Network was capable of making highly accurate classifications of attacks and normal data. After analysing the data further with PCA we discovered the ten Principle Components and found they correlated well with the data we had, and gave us insight into the working of the Neural Net. When plotting the PCA data in 2-dimensional space; we found a highly complex search space that didn't give much of a clue as to what was going on. Use of the K-Nearest Neighbours algorithm revealed promising results, but were somewhat worse compared to the MLP network. The final test that was applied was to attempt to use the PCA network as a pre-processor for the MLP network, this failed miserably.

In conclusion we discuss the results found and the importance of the design of the URL strings for the web site we are protecting. We re-iterate the success of the MLP Neural Network at providing a highly accurate classification of attacks. Finally we discuss future work in this area, in particular the use of a distributed architecture for the analysis, as well as further work on the use of Recurrent Neural Nets.

Contents

3 Introduction

The accurate and timely identification of an intruder has become of paramount importance over the past decade as the advent of the Internet has brought with it a new breed of criminals. The importance of protecting your computer system from hackers has become greatly increased as communication channels that are available both for the distribution of malicious software, and expertises in exploitation of computer systems have opened up.

With the emergence of the World Wide Web many attackers have focused their attention on the Web Server and its associated programs. This project hopes to address this problem.

Many existing systems that address the issue of Intrusion Detection tend to focus on attack signatures. When someone discovers an exploit for some piece of software, traces of the attack is left in log files. The Intrusion Detection System then looks for this "signature". There is an inherent problem with this model in that someone must manually update the database of known signatures. Of course the benefit of the system is that it will accurately identify known attacks every time. SNORT is such a system that uses this model [ids10].

Other models have included the use of rules for identifying attacks; these may focus their effort over several attacks or indeed classes of attacks. Rule based systems include the Bro system [ids6, ids5, ids8]. They are based on set of static rules which somebody decides governs an attack, again these suffer from the problem of the signature-based systems, in that they can only detect what they inherently know about already.

This project will attempt to address the problem presented by the previous two models by using a model of anomaly detection [ids4, ids9, ids11]. The anomaly detection model will be implemented using Neural Networks to analyze network traffic in real time as it passes into a Web Server. The system will attempt to identify anomalous behavior on the system and take some appropriate action.

Most Network Intrusion Detection Systems, where the network is analyzed rather than the host, look at low level protocols such as IP/TCP UDP and ICMP. This project will take an alternative approach; as it is believed that looking at these low level protocols does not always give sufficient information in determining an attack. For this reason we focus our attention on the application layer information in the hope that the semantic information available when looking at this layer will provide a clearer picture as to the nature of the packet. In particular we focus our attention on the HTTP protocol, the Web Server and the attacks which are launched over it.

The objectives of the project are as follows:

- **Determine the requirements of an intrusion detection system.** – The definition of the intrusion detection must be clearly defined for this project. Characteristics of the system must be defined, i.e. to keep a high false negative rate, to be fault tolerant [ids2] etc.

- **Evaluate the use of artificial neural networks within the problem domain.** – The use of artificial neural networks specifically in attacking the problem should be addressed, drawing on both its advantages and disadvantages over rule-based or other types of systems.

- **Determine the most appropriate type of artificial neural network to use.** – Many types of artificial neural networks currently exist; some may be more suitable for the problem than others. A detailed understanding of a number of artificial neural networks should be acquired and the most appropriate applied to the task.

- **Develop an intrusion detection system using the techniques that are found to be best suited to the problem**. – Implementation of a system will be attempted using the techniques found most appropriate.

- **Train and test the intrusion detection system using test data. –** A suitable amount of training data should be acquired and used to train and test the system.

- **Test the system on unseen data.** – A suitable amount of test data should be kept back in order to test the system's ability to recognize previously unseen attacks as intrusive behavior. This will test the usefulness of the approach.

4 Requirements Specification

4.1 Purpose

This document describes the requirements for a Neural Network based intrusion detection system. The system will be installed on a perimeter firewall installation and will monitor network traffic as it enters an internal network thus it will be a Network Intrusion Detection System or NIDS. The system will use an Artificial Neural Network to analyze the data and detect whether an intruder has attempted an attack. When the system raises the alarm the network administrator for the network should be notified and can take applicable action if necessary.

The system will employ an anomaly detection model [ids3, ids4] whereby the system is trained on normal data, so that a model of the normal usage of the server is realized within the weight matrix of the Neural Network. The system should then be capable of detecting where parameters are outside this normal model thus detecting an intruder.

The very nature of an anomaly detection model implies that it give high accurate identifications but will inherently suffer from high false positive identifications. This is because a normal system becomes very difficult to define, without traces of every possible user activity on the system. Ideally we would want the Neural Network to be capable of generalizing over the input allowing us to train it on only a small percentage of the overall search space.

We may find that the level of generalization differs with varying services offered on a system. The system is more likely to generalize over sites that contain only static web pages. More advanced pages with CGI scripts and complex database back-ends are expected to yield less accurate results.

A perfect system would be one that could have a zero false positive and zero false negative rates. However this is a difficult task when new attacks are discovered every day. The system will thus focus on gaining a low false negative rate with a reasonable false positive identification rate.

4.2 Scope

The system will be targeting the HTTP protocol on a World Wide Web Server. The System should therefore be capable of detecting attacks on the Web Server itself as well as other CGI programs installed on the system it should also be capable of detecting the exploitation of miss-configured web servers. If there is enough time Denial of Service attack detection will also be included.

The system will use three complexities of web application to test the capability of the system. The system will first be tested on a data only site, where CGI programs aren't in use, the system should cope with this scenario the best. The system should then be tested with a more complex site that has a simple form. This should be harder to detect intrusive behavior due to the variation in parameters expressed in the query strings, which are not apparent in the static model. Finally the system will be tested with a full-on e-commerce site with many CGI

programs and high database access, this scenario requires a large generalization capability of the system and is expected to yield the least accurate results.

4.3 Assumptions and Dependencies

The system will attempt to classify a particular HTTP request or an HTTP session on a Web Server as normal or anomalous. However, if users are behind a Proxy Server, accurate identifications of the session user may not be possible. Therefore for the purpose of this project we must assume that a session from a single IP address is the session of a single user.

The HTTP protocol version 1.0 specifies that a single TCP connection will carry a single HTTP message [n3]. HTTP/1.1 [n2] allows multiple messages over a single connection this system will assume we are using the HTTP/1.0 protocol.

Fragmentation provides a powerful mechanism for bypassing Intrusion Detection Systems [se7]. This system will assume these kinds of measures do not occur and that the received packet has not been fragmented. This project will not implement re-assembly routines.

4.4 Use-Case Model Survey

4.4.1 Collecting Training data

The system will come with a tool similar to tcpdump[1]. It will be told how to classify the session and produce a dump file and put it into some directory corresponding to the correct classification of the session.

Actors:

System Admin / Trainer

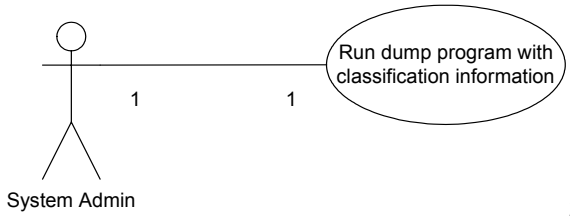


fig 4.1

4.4.2 Training Phase

The system will require a training phase in order to determine the correct weights for the neural network. This training phase will require training data that is marked as normal or anomalous. This will allow the network to adjust its weights via the use of a supervised learning algorithm to the correct values.

Actors:

System Admin/Trainer

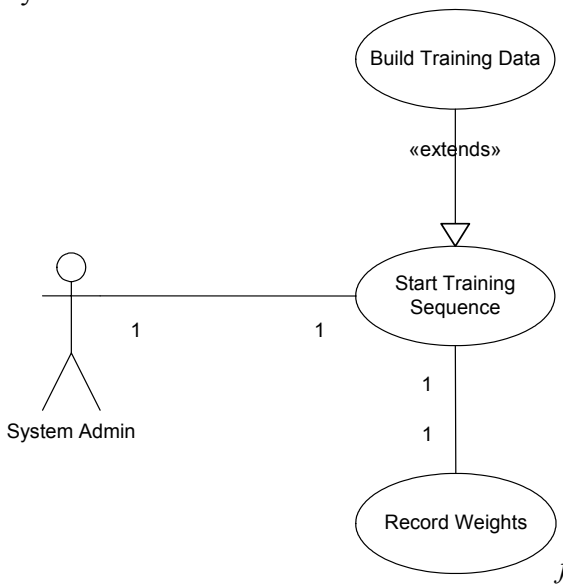


fig 4.1

[1] See glossary for description

4.4.3 Starting Service

The service will need to be started in order to begin its analysis of the network. Someone with super user privileges only must carry out this task.

Actors:

System Admin

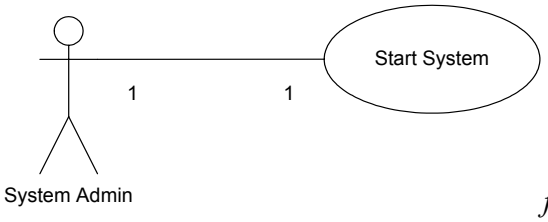


fig 4.3

4.4.4 Updating parameters

Parameters will undoubtedly need changing within the system while it is still running live on the network. Such parameters include weight matrix of the neural network and the threshold value for informing the system administrator.

Actors:

System Admin

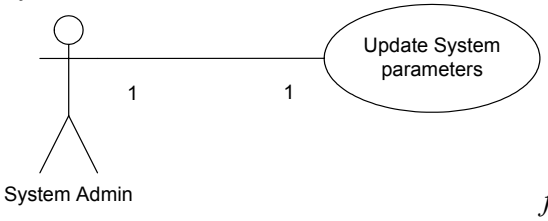


fig 4.4

4.4.5 Stopping Service

The service will need to be stopped for various reasons while it is being used. Again only a super user may do this.

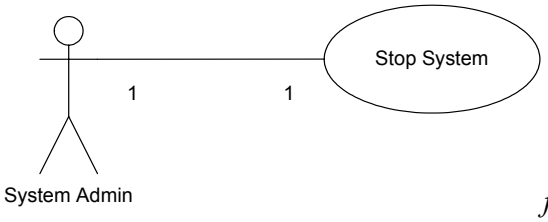


fig 4.5

4.4.6 Normal user using web server

During normal usage of the system, the system will be monitoring all network traffic bound for the Web Server that the system is protecting. In the event of normal activity the system should not respond at all and normal network service is expected. The system will keep the analyser for this client in memory for 3 minutes, so later identification of anomalous behavior is possible.

Actors:

Good User

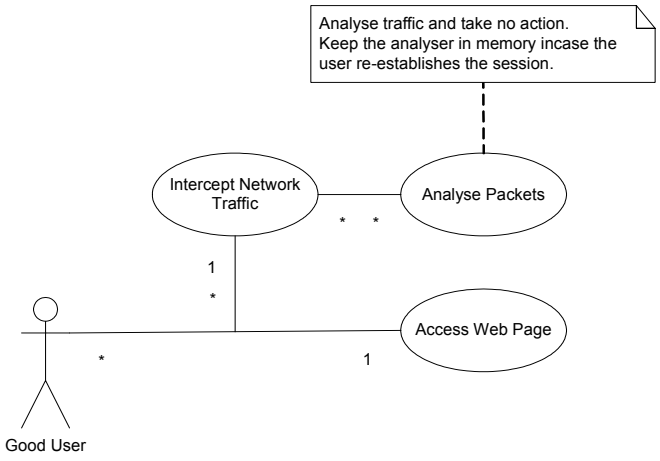


fig 4.6

4.4.7 Attacker attempting to abuse the system

At some point an alarm will be raised the neural network classifies an attack as anomalous. At this point the system admin is notified and invited to inspect the servers log files.

Actors:

System Admin
Attacker

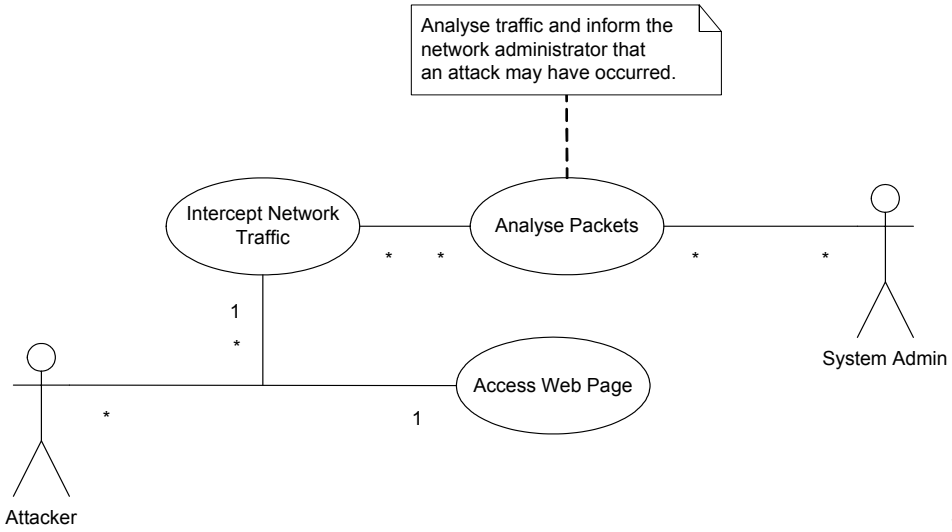


fig 4.7

4.5 Actor Survey

4.5.1 The System Administrator

The system administrator is the person who looks after the system, and is responsible for security on the system. They should be capable of determining the parameters of their system under normal operation and be capable of identifying the signs of an intruder.

4.5.2 Good User

This could be anyone who is using the web server that the system is protecting. They are simply browsing web pages.

4.5.3 Attacker

This is anyone who is trying to disrupt the services offered by the server, or is trying to gain access to the system illegitimately. This can include viruses or worm programs such as Code Red as well as humans.

4.6 Requirements

4.6.1 Functional Requirements

1. The system must be able to read HTTP messages from a variety of sources. These must include raw network packets, log files and TCP/IP sockets.

2. The user will be able collect the necessary training data using an appropriate tool.

3. The user will be able to train the neural network based analysis engine.

4. The system must be able to be started and stopped by a privileged user of the system.

5. The system will be capable of learning the difference between a normal HTTP message and an anomalous one.

6. The system is required to notify a system administrator when a sequence of events that is likely to be an attack is encountered.

7. Statistics of the systems operation must be able to be collected for analysis and review.

4.6.2 Non-Functional Requirements

1. The system must be easy to train for new users of the software. The mechanism for training the system used must be easy to understand and allow someone with a reasonable background in computer networks to capture the correct data and use the training software to create an analysis engine for their network.

2. Users of the system must be able to start and stop the system easily.

3. Users of the system must be able to configure any parameters the system may use in an easily manageable way.

4.6.2.1 RELIABILITY

System administrators should not be required to re-start the system at any point unless they wish to do so for maintenance reasons.

False negative identifications of attacks should be minimized heavily. This means a higher focus on detecting false negatives should be given to the project rather than trying to reduce false positives, although a minimized false positive rate is desirable as a secondary concern.

Intrusion Detection Systems are often themselves the target for attacks [se7], so the system should attempt to resist subversion [ids2]. So a careful, security conscious view to the development is necessary.

4.6.2.2 PERFORMANCE

Packets must be captured fast enough to enable real-time analysis of the data, and a system administrator should be notified as soon as possible.

When the service becomes slow due to heavy traffic, the system should not affect the performance of the rest of network [ids2].

The system should express a high degree of tolerance to internal faults and external faults with the system itself as well as its host system.

4.6.2.3 SUPPORTABILITY

Dynamic configuration of the system will allow the network administrators to add a different weight matrix to the system with minimal disruption to the service, and without disrupting any other services on the network.

4.6.2.4 SCALABILITY

Scalability is not really an issue in this project, however thought will be given to making this or future system able to scale up to larger identification tasks.

4.6.3 Online User Documentation and Help System Requirements

The systems API will be generated using the java documentation generator, javadoc. This will enable other programmers to develop the system further. In addition there will be a full maintenance and user manual available in html and on paper.

4.6.4 Design Constraints

The Java programming language will be used to develop the system, initially for the Linux operating system.

JPCap must be used to provide access to the libpcap c library as well as the raw sockets API. For JPCap to work libpcap must also be installed on the system.

4.6.5 Component Libraries

The JPCap library is required to implement the system. JPCap is a Java interface to the UNIX library libpcap and the Windows library winpcap. The system will be built using this library.

The Java SDK and its libraries will be required to implement the system.

4.6.6 Interfaces

4.6.6.1 User Interfaces

The user will be capable of connecting to the service in order to administer it. This will include updating parameters, starting and stopping the service. This can be graphical or text based.

The user must also be capable of training the network provided they have the data sets in the correct place for the system to find them. This can again be text or graphical based.

4.6.6.2 Software Interfaces

The system will interface with the packet capture library JPCap to enable raw packet capture and the raw sockets API.

The system will be implemented in the Java programming language and so its standard API will be used.

4.6.6.3 Communications Interfaces

An Ethernet card must be installed on the machine that is running the software. With the addition of JPCap the system will eavesdrop on the network in order to analyze its activity.

5 System Architecture

5.1 Overview

This section describes the architecture of the software system forming part of this investigation. The system has been developed as a framework allowing traces of varying formats and varying analysis tools to be written, in order to investigate the problem domain. Three sub systems have been constructed in order to logically divide the structure.

The first is the Event Engine; this provides the system with the means to read from live network interfaces/log files. The Event Engine framework will use Analysis Engine components to analyze the network traffic. The Event Engine uses a JavaCC generated parser to parse the HTTP packets that are read from raw network interfaces.

Secondly, the Analysis Engine, as previously mentioned provides a framework for the analysis of the data. It provides a set of tools with which to investigate analytical techniques used within in this project.

The final sub-system is the Main package; this contains a bunch of programs that use the Event Engine and the Analysis Engine to bring the system together. It provides a live system, conversion and dumping tools to parse raw TCP/IP packets for information and a training environment for generating performance statistics. It also contains a hand filtering tool used to hand filter packets into separate files.

Essentially we are creating a suite of tools that will collect, decode/encode, and analyze a data set containing both normal data and attack data. The suite of tools is such that we can configure it to read from a number of sources, encode the data in a number of ways and analyze the data using a variety of techniques.

The use of these tools flows as follows;

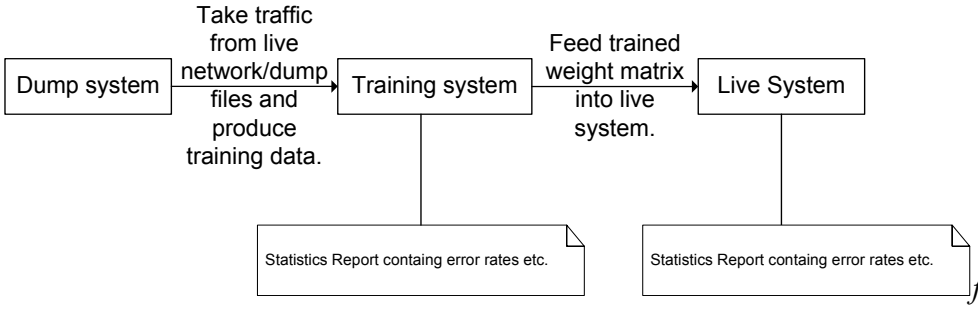


fig 5.1

Each of these tools will use parts from both the Event Engine and the Analysis Engine. Each package collaborates with the following dependencies;

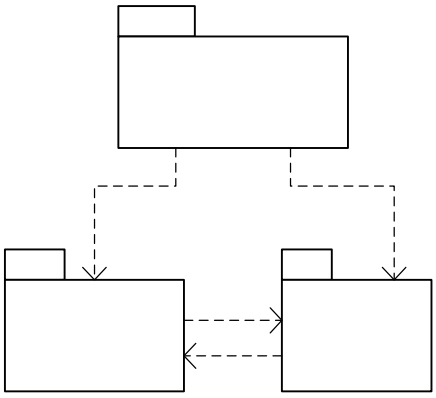


fig 5.2

The Analysis Engine uses some of the event engine classes in order to analyze the data in an implementation independent way. The Event Engine must be aware of the analyzer that is used to analyze the network traffic and so this dependency also exists. Of course the Main package WebIDS must be dependant on both the Analysis Engine and the Event Engine for its operation.

5.2 WebIDS.EventEngine

5.2.1 Commentary

The Event Engine package contains the functionality and logic with which to read HTTP requests from some source in an independent manner, constructing an appropriate representation for this source. This data is then simply passed up to the Analysis Engine for analysis. The Event Engine will react accordingly to the analysis of the HTTP request by taking appropriate action. The action that will be taken is dependant upon the configuration of the system, i.e. whether log files or raw packets are being read.

5.2.2 Static Structure

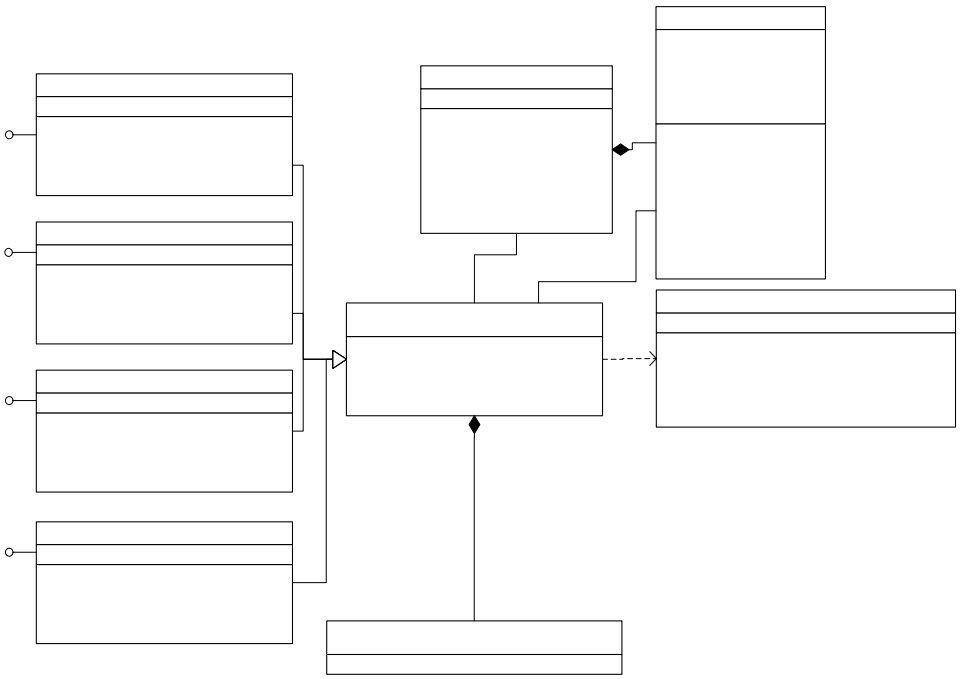


fig 5.3

5.2.3 EvReader and implementations

The EvReader interface provides the necessary interface with which to train and run the system in a live state.

void	**addObserver**(EvObserver obs) allows you to register an object as an observer, so that appropriate actions can be taken when analysis has been conducted.
void	**build**(java.lang.String fname, EvTrSet tset) builds an internal representation of the resource corresponding to filename.
void	**interruptTimer**(long millis) allows the user to interrupt either of the two other operations.
void	**run**(java.lang.String fname, AeAnalyser analyser) runs the resource corresponding to filename in a live state.
void	**setThreshold**(double t) allows you to set the threshold for determining good and bad packets.

There are several implementations of this interface in the system each of which provides the desired functionality whilst reading from a number of different sources.

EvReaderLogFile

This class is an implementation of the EvReader interface which can read HTTP requests from apache log files. It builds the training set by reading the log files from disk.

EvReaderJPCap

This class implements the EvReader interface by using the packet capture library JPCap which is a JNI interface to the infamous unix packet capture library libpcap. This reader is an implementation of a Packet Sniffer therefore it allows the user of this reader to read raw packets either Live on the network or from dump files.

EvReaderSocket

This class is an implementation of the EvReader interface which reads the HTTP requests by listening on a socket. After valid analysis this class will forward the request onto the web server. Therefore this class is acting as a Proxy Server.

EvReaderText

This class is an implementation of the EvReader interface which reads the HTTP requests from the AeAnalyser dump file format. This is a comma delimited format.

5.2.4 EvTrSet

The EvTrSet class contains a bunch of HTTPRequest objects. It provides methods to access these objects in an iterative fashion. It also provides the method to convert these objects into input vector representations.

void	**add**(EvHTTPRequest hm) adds an HTTP message to the t set.
void	**addRandom**(int count) generates a load of random HTTP requests.
EvHTTPRequest	**get**(int i)
double[]	**input**(int i)
int	**size**()
EvTrSet	**splitSet**(double prob) splits this training set up into two sets.
double[]	**target**(int i)
java.lang.String	**toString**() returns a string representation of the trset.
void	**truncate**(int ncount) truncates the input to size ncount

The EvTrSet class contains extra functions to enable manipulation of the training sets, example of such functions are splitting the sets to implement cross validation, as well as truncation, that removes any stored requests past the specified count.

5.2.5 EvHTTPRequest

The EvHTTPRequest object represents an HTTP request; it should contain all the information available to the original source. If the request was built from a raw packet, then it should contain every part of the HTTP header as well as a destination/source IP address etc. The EvReader objects build these objects from their own data files. They can be added to EvTrSet's in order to be used in analysis training or simply analyzed individually.

java.lang.String	**getMethod**()
java.lang.String	**getURI**() returns the uri string for this request.
java.lang.String	**getVersion**()
double[]	**inputVector**() This method constructs an input vector suitable for analysis by the analysis engine.
boolean	**isAttack**() returns whether this packet is flagged as being an attack.
java.lang.String	**key**()
void	**setAttack**(boolean fl) sets whether the packet is an attack or not.
void	**setMethod**(java.lang.String m) allows a user of this class to set the method
void	**setURI**(java.lang.String uri) allows a user of this class to set the requesturi variable.
void	**setVersion**(java.lang.String vs) allows a user of this class to set the version variable
java.lang.String	**toString**() Turns this object into a string representation will be used when dumping logs to disk etc.

5.2.6 ParseHTTP JavaCC grammar

This is not a java file, but a javacc file. It contains a grammar that is compliant with the HTTP/1.1 RFC [n3].

This was a direct translation from the B.N.F. specified in the RFC. EvReader objects use this parser to parse their HTTP requests and build EvHTTPRequest objects. The parser constructs the packet retaining information such as the URL string, the request method, and the header information.

5.3 WebIDS.AnalysisEngine

5.3.1 Commentary

The Analysis Engine contains the logic for the analysis of the data provided by the Event Engine. The Analysis Engine consists of an interface AeAnalyser, this interface is implemented by five classes which perform some analysis of the data in order to classify the HTTP request objects as attacks or not.

The Event Engine is provided with the result of the analysis and is prompted to take appropriate action depending on the result of the analysis. The Analysis Engine is designed upon the assumption that the AeAnalyser classes will require training, thus some form of training algorithm must be implemented.

5.3.2 Static Structure

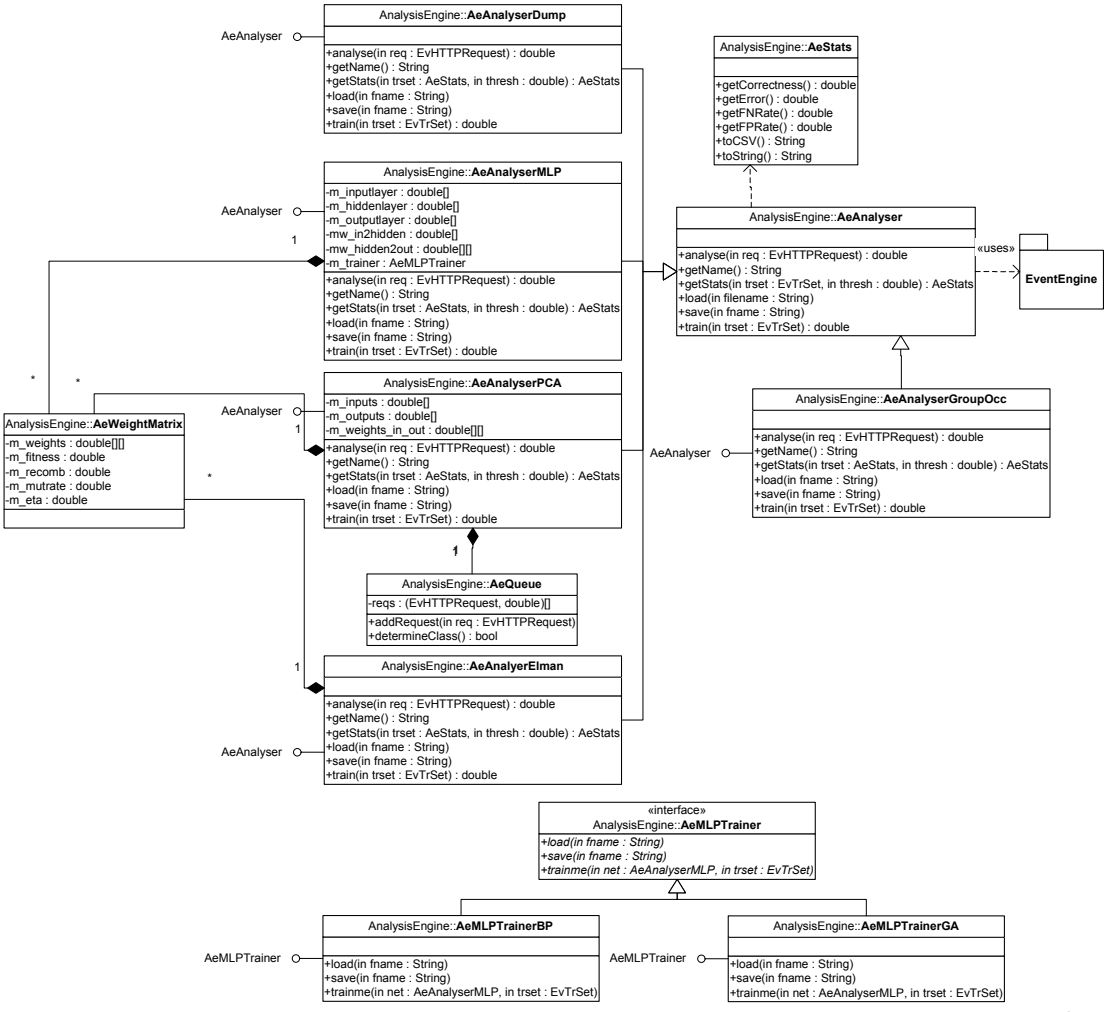


fig 5.4

5.3.3 AeAnalyser interface and implementations

The AeAnalyser interface provides the Analysis Engine framework with an independent way for several data analysis techniques to be used by the Event Engine. AeAnalyser objects may be registered with an EvReader, when a reader needs to analyze some data it will used its registered object.

double	**analyse**(EvHTTPRequest httpreq) perform some analysis of this httpreq object.
java.lang.String	**getName**()
AeStats	**getStats**(EvTrSet trset, double threshold)
void	**load**(java.lang.String file) loads analysers internal structure from disk.
void	**save**(java.lang.String file) saves analysers internal structure to disk.
double	**train**(EvTrSet trset) train this analyser with this data.

There are five implementations of this interface used in this investigation;

AeAnalyserDump

This is an implementation of the AeAnalyser interface. It is simply designed to convert an EvHTTPRequest object into a text representation in a custom log format for inspection later, or simply for converting the data into a format that can be used at a later date.

AeAnalyserPCA

This class is an implementation of AeAnalyser which implements Principle Component Analysis (Section 6.3). It logs the results of the PCA and then performs the K-Nearest neighbors returning the class with the largest match. The PCA is implemented as an unsupervised Neural Network using the Oja's rule [nn3].

AeAnalyserMLP

This class is an implementation of AeAnalyser which implements a Multi-Layer Perceptron Neural Network [nn3, nn0, nn4] (section 6.2). This Neural Network may be trained with a number of algorithms and so contains an implementation of the interface AeMLPTrainer, namely a back-propagation type algorithm and a Genetic Algorithm.

AeAnalyserGroupOcc

This class implements the AeAnalyser interface with a simple analysis of the frequencies of groups of letters within the URL string, i.e. the number of vowels or locator characters.

AeAnalyserElman

AeAnalyserElman is an implementation of the AeAnalyser which uses an Elman network [nn1] (section 6.4.2). An Elman network is very similar to a MLP network except it has minimal recurrent links. Therefore allowing it to find structure in time over all the requests it has seen before.

Due to its similarity to MLP networks, the same Back Propagation and Genetic Algorithms can be used to train it. The Back Propagation on an Elman network is called Back Propagation through time [nn2].

5.3.4 AeAnalyserMLPTrainer and implementations

AeMLPTrainer is an implementation of the design pattern Command [r1]. The trainme() method in the interface passes in the training set as well as the network to be trained. It also contains methods for loading and saving internal state if the training algorithm has any.

The an AeMLPTrainer implementation may be registered with an AeAnalyserMLP class and when the train() method is called the trainme() method of the registered AeMLPTrainer object is called.

void	**load**(java.lang.String str) allows the user to load any internal state.
void	**save**(java.lang.String str) allows the user to save any internal state.
void	**trainme**(AeAnalyserMLP net, EvTrSet trset) given a neural net, will perform some kind of training function

There are currently two implementations of this interface;

AeMLPTrainerBP

This class implements the Back Propagation algorithm with a momentum term (section 6.6).

AeMLPTrainerGA

This class implements a two individual tournament selection Genetic Algorithm (section 6.6).

5.3.5 AeStats

This object provides the users of this package with some statistics information in order to determine the performance of the system. It contains information such as number of requests gone through and how many have been misclassified as well as rates for false positive and false negative identifications for later analysis.

double	**getCorrectness**() returns the correctness rate of the run being analyzed. For example a perfect system would be 100%. If it failed on every identification it would be 0% correct.
double	**getError**() returns the error normally as the sum of squares (section 6.6).
double	**getFNRate**() returns the percentage of false negative identifications.
double	**getFPRate**() returns the percentage of false positive identifications.
java.lang.String	**toCSV**(java.lang.String label) converts the stats into a CSV format with a label.
java.lang.String	**toString**()

5.3.6 AeQueue

AeQueue is an implementation of a priority queue, however if there is not enough space the ones at the bottom are dropped. It used to determine the k-nearest neighbors (section 6.7). The identification which is most prominent within the queue is chosen as the correct identification for a new request. The score which a request is added with represents the Euclidean distance between the request we are attempting to classify and another request in the system already.

void	**addRequest**(EvHTTPRequest hr, double score)
boolean	**determineClass**() From the requests in the queue which class is in the majority.

5.4 WebIDS main package

5.4.1 Commentary

The Main WebIDS package is the responsible for providing the interface and interaction to the human user of the system, namely the system administrator. A suite of tools is provided for training, tuning and experimenting with different settings as well as producing statistics for analysis with other appropriate tools, of course the suite also comprises of a live system in order to deploy the system in a production environment.

5.4.2 Static Structure

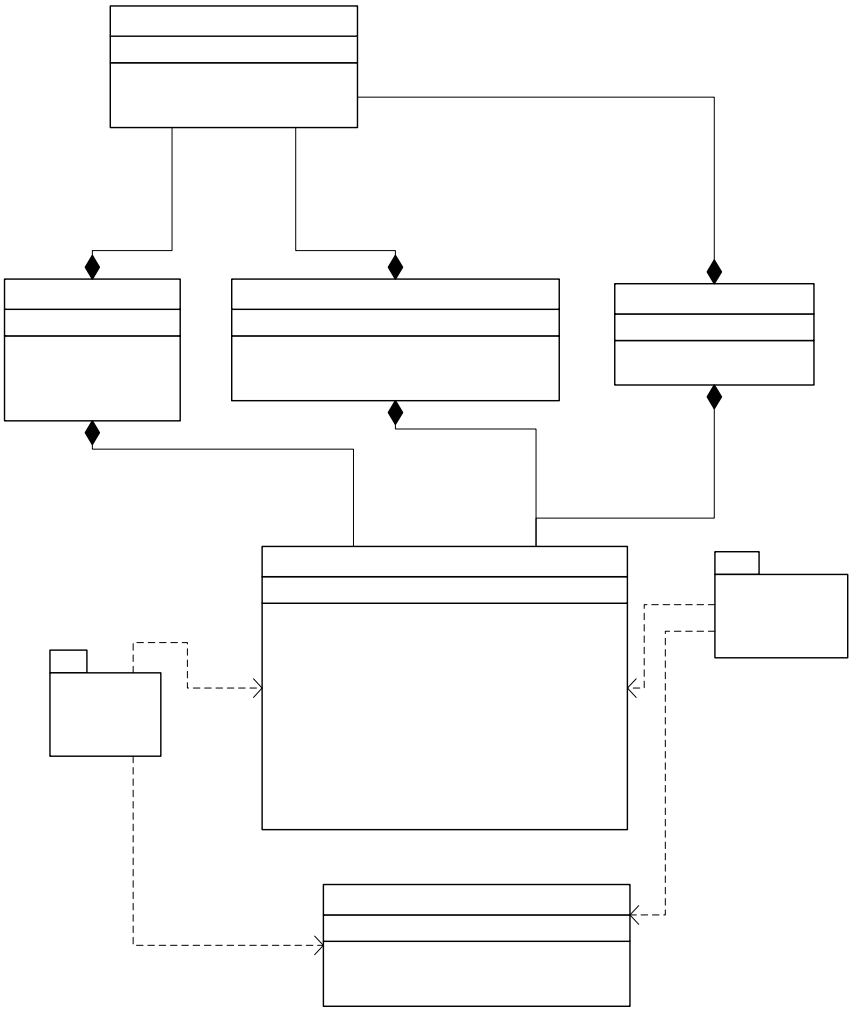


fig 5.5

5.4.3 WiComponentFactory

This class is an instance of the Factory design pattern [r1]. It reads from configuration files and allows the rest of the system to be built in an implementation independent way. It is also an instance of the Singleton design pattern [r1].

static WiComponentFactory	**instance**()
AeAnalyser	**makeAnalyser**() makes an AeAnalyser object.
EvReader	**makeReader**() makes a EvReader object

The methods makeAnalyser builds an AeAnalyser object according to the implementation which it is currently configured to use. The makeReader method also builds an EvReader according to the currently configuration settings.

5.4.4 WiUtils

This class provides a set of utilities for the entire system to use.

java.lang.Object	**arrayGrow**(java.lang.Object a) makes an array double in size, keeping the class type etc.
int	**countGroupOcc**(java.lang.String xx, char[] group)
double[]	**frequencyAnalysis**(java.lang.String str) analyses a string looking at frequency of characters.
void	**infect**(double[][] infector, double[][] infectee, double recomb, double mutationrate, double eta) infects one matrix with another.
void	**initWeights**(double[][] wts) randomly initialises weights in a matrix
java.lang.String	**inputVectorToString**(double[] ipvec) Simply prints out a double array representing an input vector.
static WiUtils	**instance**() returns the single instance of this class.
int[]	**randomArray**(int max, int all) returns an array of max numbers picking from between 0 and all. The array will contain no duplicates.
java.lang.String	**randomString**() returns a random array corresponding to a random string.
double	**sigmoid**(double x) implementation of the sigmoidal function 1/e(-x).
double[]	**urlToIV**(java.lang.String uri) given a url string converts it into a binary representation of a url string.

5.4.5 WiLogFile

This class represents a log file on the disk. When the constructor is called the filename of the log file is stored. Sub-sequent writes and reads from the object refer to the file specified in the constructor.

	WiLogFile(java.lang.String fn) constructs a LogFile associated with a particular file.
java.util.ArrayList	**readLog**() reads the contents of the log into an ArrayList.
void	**writeLog**(java.lang.String message) writes a string to the log file.

5.4.6 WiOpenLive

This class provides a main method for the system to be opened in a live state. If we were using the JPCap reader that would mean opening up the Ethernet card in promiscuous mode, if it were a Proxy Server it would mean opening up the TCP/IP server. The WiComponentFactory is used to create the EvReader objects and AeAnalyser objects.

void	**buildFromDir**(java.io.File dir) runs the m_reader to build the trset.
static void	**main**(java.lang.String[] args) opens the interface live.
void	**openLive**(java.lang.String fname) opens the read live, with the file/dev given int he params..
void	**runSet**(java.lang.String dir) runs the reader on a directory..

5.4.7 WiTrainEnv

This class is the environment for training the AeAnalyser that is currently registered as the default analyzer. It will build the training sets from the command line arguments, build the training sets and call the train() method for the analyzer.

void	**buildFromDir**(java.io.File dir, EvTrSet trset) runs the m_reader to build the trset.
EvTrSet	**buildSet**(java.lang.String dir) builds a training set given a directory, reads up all the files in the directory and parses them for dumps.
static void	**main**(java.lang.String[] args) main method runs the trainer..
void	**train**(java.lang.String dir) trains the Analyser

5.5 Collaborations

The main system will run in essentially two modes. The first is used to generate an appropriate weight matrix for the Analysis Engine namely a Neural Network. The second phase is to open up the system in a live state. Because the trace data and indeed the live system will analyze data from a number of sources, it is necessary to implement the EvReader interface a number of times in order to accommodate for the variety of data sources being used. Equally the AeAnalyser interface will be implemented a number of times in order to experiment with a number of different analysis techniques.

Because of the abstract nature of the systems framework these collaborations deal with the interfaces used. All implementations will fit into this general framework.

5.5.1 The live system framework Sequence Diagram

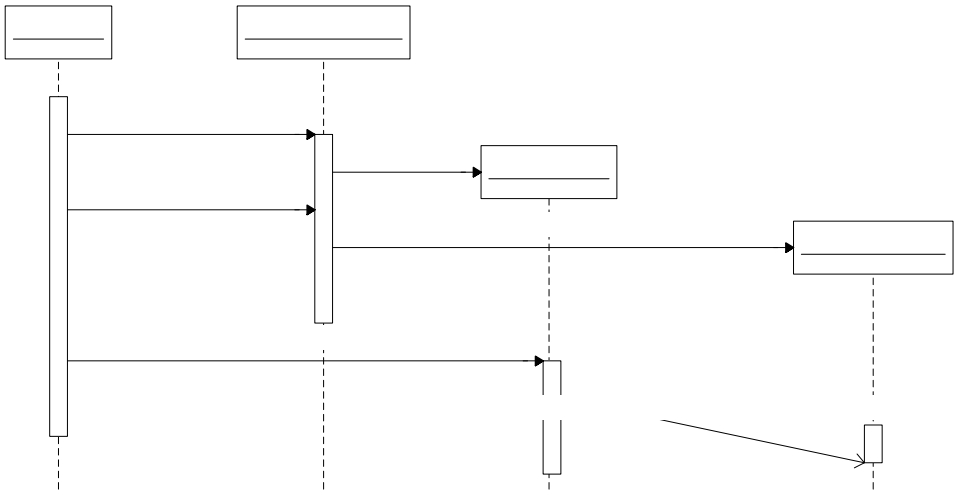


fig 5.6

Fig 5.6 shows the interaction between the main objects and interfaces within when the live system is started. The Component factory builds both the AeAnalyser and EvReader classes and provides the WiOpenLive class with a set of methods for opening a user specified interface.

Of course the actual implementations of the EvReader and AeAnalyser classes are not shown here due to their complexity. This diagram shows how the framework interacts, see the Underlying Principles section (6) for the algorithms used within the framework.

5.5.2 The training system sequence diagram

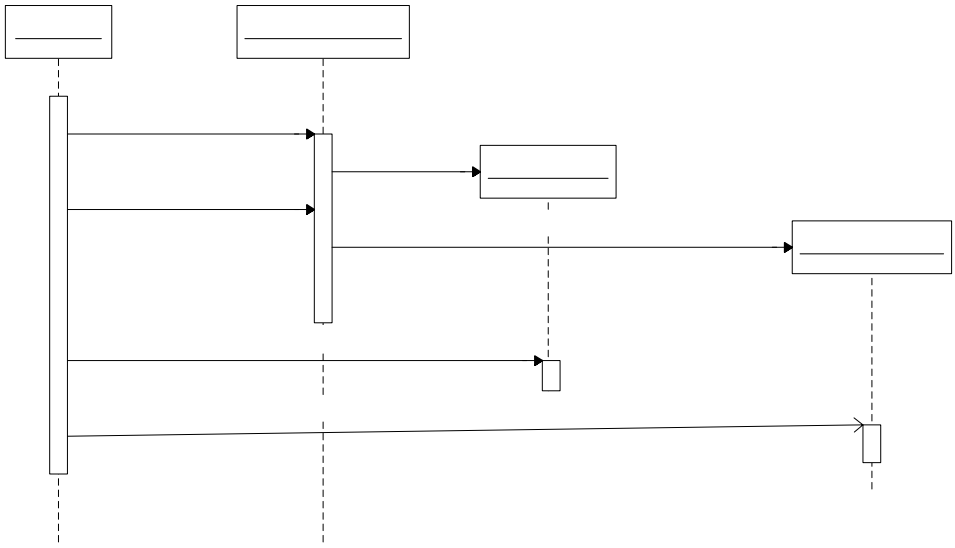


fig 5.7

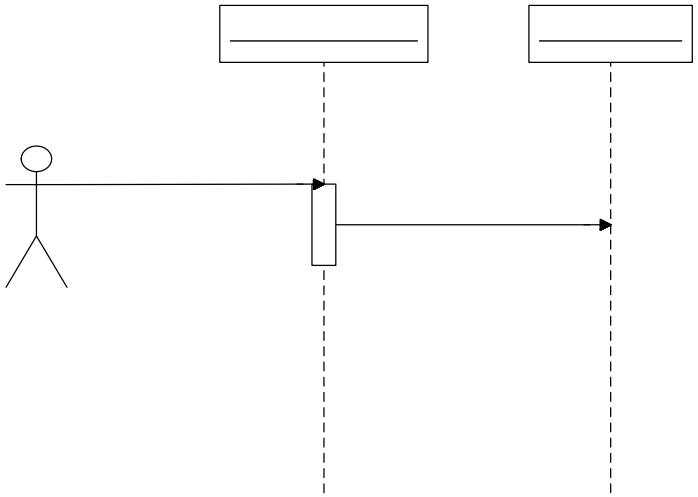
The training system is similar to the live system interaction except the WiTrainEnv retains control after the EvReader object has read the necessary trace data. Notice also that the EvReader is no longer used after this point. The training can then begin by repeatedly calling train on the AeAnalyser object.

:WiTrainEnv

makeReader:

makeAnalyser:

5.5.3 The Multi-Layer Perceptron training algorithm framework



This is an implementation of the template design pattern. Which ever trainer is currently registered with the AeAnalyserMLP object is called via its trainme() method.

net:Ae

train:=train(trset)

User

6 Underlying Principles

6.1 Motivation for Analysis Framework

6.1.1 Motivation for looking at Application Layer

Automatic identification of intrusive and malicious behavior on computer networks is becoming an increasingly difficult task. Most existing systems use information available at the lower level layers of the TCP/IP protocol suite such as IP, TCP/ICMP/UDP. This of course gives these systems a great deal of power in identifying attacks on networks; IP spoofing [ids8] TCP flooding and other similar attacks on the network and transport layers may only be detected by inspecting this level of the protocol stack.

However when we take a look at attacks which occur at the application level we find there may not be enough information available at the lower levels. TCP segments may look completely normal whilst containing malicious code. This motivates the investigation of a specific application in attempt to determine whether the semantic information available at the application layer will give us more insight into the identification of the attack. For this project the HTTP protocol will be used to investigate the implications of the approach.

Previously we mentioned the use of the Anomaly detection model [ids4, ids9, ids11]. This model fits quite well when we use this approach to the analysis of the data. The normal usage of the system can be modeled, by applying some statistical modeling technique and breaches to this "normal" may be detected by determining whether the new data fits within normal parameters.

6.1.2 A closer look at the HTTP protocol

HTTP (Hyper Text Transfer Protocol) is now the most popular Internet service [n2]. The increasing use of the Web has however encouraged its abuse. An increasing number of exploits for the Web Server and its associated programs are being discovered every day.

We as system administrators can determine attempts to compromise our system normally by simply inspecting our log files. It becomes very clear when we look at our "normal" traffic that a particular request is not part of a normal user's activity. Log files normally contain information such as the request method, the URL and the response the web server gave.

If we take a look at some log entries for some normal web sites the point above is made clear;

Buildersdirect.co.uk

```
212.241.139.4 GET /default.asp?qtbid=5&qtcid=QL&qtqecid=49  200
212.241.139.4 GET /Brands/BuildersDirect/scripts/GenralUtility.js  200
212.241.139.4 GET /QuoterEngine/inc/stdlib.js  200
212.241.139.4 GET /QuoterEngine/inc/validate3.js     200
212.241.139.4 GET /default.asp?qtbid=5&qtcid=QL&qtqecid=49  200
192.168.0.22  POST /default.asp?qtbid=5&qtcid=QL&qtqecid=29 200
```

Fkellis.com

```
212.241.139.4 GET /fkellis/logo.gif 200
212.241.139.4 GET /fkellis/index.html 200
212.241.139.4 GET /fkellis/tap.jpg 200
212.241.139.4 GET /fkellis/about.html 200
```

These are a few requests made from the buildersdirect.co.uk and fkellis.com web servers respectively. Let's look at the information which logs like this give us;

Firstly they give us the IP address that the request came from; this is useful because we may be able to tell where the request came from. Of course there are many ways to hide the origin of the attack, by spoofing the IP Address [ids4] or going through a proxy, or simply using a different machine from your own. Clues can be discovered from the IP Address; the Code Red worm tends to attack hosts with very similar IP Addresses to the originating hosts, this could give a clue as to the intent of the request. This avenue of investigation seems less important in determining the presence of intrusive behavior as it tells us only the origin of the attack.

Secondly let us look into the request type. Of course this is very important, are we using GET, POST HEAD etc. This is an important piece of information to look at. However it complicates the investigation somewhat and therefore the investigation will be restricted to GET request types, most CGI attacks use GET anyway.

Thirdly the URL is given; this gives us a very large amount of information. It tells us the resource that the request is asking for as well as any parameters that may be passed to the resource identified by the URL. This gives an attacker access to a particular resource which may use some code which is exploitable. We find that most attacks will use a carefully crafted URL string in order to exploit security holes in the resource;

Directory traversal attacks of various kinds are exploitable when a system does not apply the correct permissions of the directory where their scripts are run. This allows an attacker to traverse the directory structure, and execute commands of their choice. Attackers will use these kinds of attacks in order to execute general purpose interpreters such as cmd.exe, bash, csh, ksh etc; As well as to grab password files for cracking later.

Examples of this are below;

```
GET /cgi-bin/PRN/../../../../../../../../WINNT/system32/ipconfig.exe HTTP/1.0
GET /sojourn.cgi?cat=../../../../../etc/password%00 HTTP/1.0
GET /a1stats/a1disp3.cgi?/../../../../../../etc/passwd HTTP/1.0
```

There are many variations on this kind of attack including replacing the / with the Unicode equivalent, as the infamously insecure web server IIS implements a poor scanning facility for this kind of attack. The Nimbda worm exploits many of these vulnerabilities;

```
GET /scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir HTTP/1.0
GET /_vti_bin/..%255c../..%255c../..%255c../winnt/system32/cmd.exe?/c+dir HTTP/1.0
GET  /..%25%35%63../scripts/cmd.exe?/c+dir HTTP/1.0
```

Other attacks include buffer overflow attacks [$x], where an attacker attempts to write arbitrary code into the memory of some program, causing a stack/heap smash resulting in the execution of arbitrary code. These kinds of attack are very obvious when we see them in a log;

```
GET /default.ida?XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%25u9090%25u6858%25ucbd3%25u7801%25u9090%25u6858%25uc
bd3%25u7801%25u9090%25u6858%25ucbd3%25u7801%25u9090%25u9090%25u8190%25u00c3%25u0003
%25u8b00%25u531b%25u53ff%25u0078%25u0000%25u00=a HTTP/1.0
```

We can instantly see by looking at the code red worm a few oddities that indicate its malicious nature. The first and obvious thing to notice is the large number of X characters; these are used to locate correct memory location to dump the rest of the code into the exploited programs memory space. We also notice a largely abnormal number of Unicode and hexadecimal characters which we don't tend to see in the normal traffic. This is the buffer overflow egg and contains machine instructions which will be executed if the exploit is successful.

Other suspicious URL strings contain commands such as the UNIX "cat", various redirects and pipe characters (> |), such as this URL, this will return the contents of the password file;

```
GET /cgi-bin/simple/view_page?mv_arg=|cat%20/etc/passwd| HTTP/1.1
```

The final piece of information that the logs give us is the response code. The response code expresses how the Web server dealt with the request. It is useful to know if people have been searching about for files that don't exist; or have successfully obtained files, so the response code is useful for auditing after the attack has happened. However in the context of an early warning system the response code is not known unless the system stores some state. This makes the implementation over complicated and demands the passing of packets before deciding whether they contain attacks or not.

6.1.3 Developing the URL scanner

Now we have established the great significance of the URL string in the identification of abnormal activity on the Web Server it becomes clear that we need some way to automatically scan the URL for abnormal or malicious intent. The first thing that comes to mind is the equality matching or some kind of more advanced pattern matching routine for identifying known exploits. This kind of thing is fine and there is no reason why it should not work; it in fact implements the misuse detection model [ids1, ids4].

One problem with this model is the maintenance of the signatures database. Of course new vulnerabilities are discovered every day and the list will become larger and larger. This increase in size will also bring about a decrease in speed as more patterns will need to be checked.

This could potentially be solved by using some kind of complex function which can check the URL string against a variety of patterns in one pass. Artificial Neural Networks provide a solution. The issue of database maintenance can be solved trivially by implementing the anomaly detection model [ids4, ids9, ids11] and defining the systems' normal URL strings and allowing the system to determine deviations from this normal.

The final problem to solve now is how to encode these strings into the Neural Network. The discussion in section 6.2 outlined a few types of attack against a system. It was outlined how, by eye, we can tell which URL string is malicious and which is not. So what do we see in the attacks that we don't see in the normal URL strings?

It is proposed that the indication of normal data is through its substrings. One websites' set of URLs seem to contain very similar sub strings within them. This leads on to the actual frequency of each character occurring in the string.

One way to encode this would be to use zip encoding [$x]. This, however, would require either a very large set of inputs or a lot of preprocessing on the data before it entered the Neural Network. An alternative would be to encode the URL string as simply the normalized frequencies of the characters within it. For example;

The string `/index.html` would look like this (omitting the characters that have zero values);

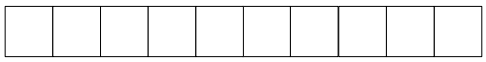


fig 6.1

The string for the code red worm looks like this again omitting zero valued characters;

```
/default.ida?XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXX%25u9090%25u6858%25ucbd3%25u7801%25u9090%25u6858%25ucbd3%25u7801%25u909
0%25u6858%25ucbd3%25u7801%25u9090%25u9090%25u8190%25u00c3%25u0003%25u8b00%25u531b%2
5u53ff%25u0078%25u0000%25u00=a
```

fig 6.2

This encoding is accompanied with a binary encoding of the length of the string so for example `/index.html` would also include in the encoding 00001010 on the end. A Neural Network should be capable of using this information to build a statistical model of the system using the provided information. This is the encoding and structure that will be used for the rest of the investigation as it seems like a good representation of the data and hopefully will provide us with high dispersion in the data sets that are used.

.00	.00	.00	.00
3	7	3	3
a	d	e	f

6.2 Multi Layer Perceptron (MLP) trained with Back Propagation and Genetic Algorithms

6.2.1 What is a MLP?

The Perceptron is a single unit pattern recognition machine invented in the 50's by Rosenblatt. He derived an algorithm for finding appropriate weights in a finite amount of time such that any linearly separable pattern could be recognized [nn3]. The Multi-Layer Perceptron (MLP) extends the original Perceptron by including a hidden layer of processing units. The first layer of units feed into the second layer and second into the third layer of processing elements. This new structure is capable of solving a problem which is non-linearly separable [nn0, nn4, nn3]. The XOR problem is an example of a non-linearly separable problem, the space cannot be divided by a single hyper-plane (line in two dimensions), but requires a number of hyper-planes (see glossary).

A typical MLP network is shown fig 6.3, a bias is attached to layer which is set to 1. This controls the point at which the sigmoid function is activated and its weight can be trained along with the standard training procedure.

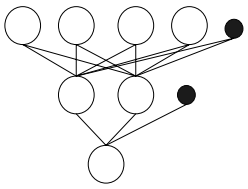


fig 6.3

The output of a three layer network with d input units, m hidden units and c output units with a bias node on the first and second layers can be expressed as;

$$aj = g\left(\sum_{i=1}^{d} \text{ẅ}ji^{(1)}\, xi + \text{ẅ}j0^{(1)}\right) \quad \textit{(eq 1).}$$

Where *aj* is the input to the hidden unit *j*, $\text{ẅ}ji$ is the weight between the *i*th input unit and *j*th hidden unit, and *xi* is the input from the *i*th input unit. The extra term $\text{ẅ}j0^{(1)}$ is the bias node and is always clamped as *x0 = 1* [nn0].

The function g is a sigmoid function which is normally expressed as;

$$y = 1/(1-e^{-x}) \quad \textit{(eq 2).}$$

The output from the network can be similarly expressed as;

$$ak = g\left(\sum_{j=1}^{d} \text{ẅ}kj^{(2)}\, xi + \text{ẅ}k0^{(2)}\right) \quad \textit{(eq 3).}$$

Where $w_{kj}^{(2)}$ expresses the weight between the *j*th hidden node and the *k*th output node, a*k* represents the input to the *k*th output unit. Again the function g is normally the sigmoid function expressed in eq 2.

These equations are implemented in the AeAnalyserMLP class and provide an implementation of AeAnalyser such that the analysis of the EvHTTPRequest objects may be conducted using this technique.

6.2.2 Training Algorithms for MLP Networks

The main function of a these types of Neural Network is the algorithms available for finding an appropriate weight matrix which will solve the problem you are wanting to solve. This means the inevitable use of an algorithm for realizing the weights of the network. In this project two such algorithms are considered. The first is a tournament selection based Genetic Algorithm, the second based on the original Back Propagation with a momentum term added [nn3, nn0]. Momentum helps resolve the problem of becoming stuck in local minima and provides smoothing of the weight space [nn0].

The Genetic Algorithm is a very simple two member tournament based selection algorithm with creep mutation [nn5]. It is specified below;

```
Algorithm runAlgorithm(TrainingSet ts)
Begin
        Candidate1 = random member of population
        Candidate2 = random member of population

        If (candidate1.fitness>candidate2.fitness) then
                candidate1 ↔ candidate2

        foreach w ∈ candidate1.weights
                if (probability of recombination)
                        candidate2.weights[w] ← candidate2.weights[w]
                if (probability of mutation)
                        candidate2.weights[w] ← candidate2.weights[w]
        end

        candidate2.fitness ← fitness(candidate2)
end

Algorithm fitness(member)
Begin
        Sum = 0
        Foreach t ∈ ts
                Networkweights ← member.weights
                Fire(ts.getInputs(t))
                Sum+=Calculate error2
        end
end
```

fig 6.4

The Back Propagation algorithm is specified as follows;

```
Algorithm runAlgorithm(TrainingSet ts)
Begin
        Foreach t ∈ ts
                Fires(ts.getInputs(t))
                Foreach oNode ∈ outputlayer
                        Error ← target(t) – output(t)
                        Calculate Delta for o
                        Foreach hNode ∈ hiddenlayer
                                CalculateWeightDeltas
                                AdjustWeights
                        end
                end

                Foreach hNode ∈ hiddenlayer
                        error ← 0.0;
                        Foreach oNode ∈ outputlayer
                                Error = delta for o * wts_h_o[h][o]
                        end

                        Foreach iNode ∈ inputlayer
                                CalculateWeightDeltas
                                AdjustWeights
                        end
                end

        end
end
```

fig 6.5

Both algorithms have the same theoretical running time of O(n) [nn5]. Both algorithms have other similar credentials; both algorithms are valid training algorithms for MLP networks. In general the Back Propagation algorithm tends to be more reliable at finding a similar accuracy in each run of the algorithm.

These algorithms will be implemented within the AeMLPTrainer interface allowing them to "plugged" into the framework of the system and allowing their performance to be compared.

6.2.3 Application of MLP's to the problem domain

The system will incorporate the anomaly detection model [ids4, ids11]. This model specifies that the system be "taught" in some way that a bunch of inputs is normal. Variations from this normal model can then be detected using an appropriate technique. Of course MLP Neural Networks are capable of learning any function, and so could potentially learn the pattern which warrants "normal" data.

The system that will be used for training the network is specified below;

```
Algorithm anomalyDetectionTraining()
Begin

        TrSet set1 = getTrainingSet()
        TrSet set2 = set1.split() // this will break the training set into two equal parts.

        NeuralNet n;

        For i=0 to epochs
                n.train(set1)

                n.testQuality(set1)
                n.testQuality(set2)
        Next
End

(part of MLP implementation class)

Algorithm testQuality(TrSet)
Begin
        Foreach t ∈ set1
                output ←fire(t)

                // Error is implemented as sum of the squared
                // error of the outputs against the normal
                // target.
                if (Error(output) > 0.5)
                        raiseAlarm()
        Next
End
```

fig 6.6

6.3 Principle Component Analysis Networks

6.3.1 General discussion of PCA

Principle Component Analysis is a well known statistical technique [nn3]. It is the process of taking some high dimensional input data and projecting it onto a lower dimensional space whilst maximally preserving the information about the original input data. It is a type of feature extraction [nn0] where we reduce the complexity of the input data to gain increased speed and generalization capabilities of the Neural Network.

There is however other reasons for applying PCA to data excluding feature extraction. This may include actually analyzing the vector which represents the Principle component, it can tell us a lot about the data. The first Principle Component provides us with the vector which gives maximum variance in the data. Of course sub-sequent Principle Components are the vectors where the data is projected ortho-normal to the previous components which give maximum variance in the new projected space.

Implementations of PCA may use matrix equations [nn0]. However there are alternative ways to perform PCA, one such way is to use an unsupervised Neural Network [nn3]. This project will investigate the use of Principle Components on the data in order to learn more about the data in the chosen encoding.

6.3.2 Associative Memories and Oja's rule and Sangers rule

Associative memories work by strengthening a connection each time it is used. This results in the weights which are used more often to become stronger. In the context of an Artificial Neural Network, we can see that patterns the Network has seen previously are more likely to give a higher response then a pattern that has not been seen before.

Of course several things need to be taken into account when using such a technique, as if we simply continue to increase the weights they will become very large. Many people have developed algorithms for the normalisation of these kinds of Networks namely Oja [1982].

$$w_i(n+1) = \frac{w_i(n) + \eta y(n) x_i(n)}{\sqrt{(\sum(w_i(n) + \eta y(n) x_i(n))^2)}} \quad \text{(eq 6.1)}$$

This equation will allow a bunch of weights to be updated so that the inputs seen most often will allow the Processing unit to be given a higher value. Using this type of Neural Network we are able to implement PCA. The equation in 6.1 finds the maximum variance among the input data. This is of course the first Principle component. Using a slightly modified version of this equation we are able to apply the deflation method [nn3]. This will project the data onto an axis perpendicular to the first eigenvector, allowing us to apply the algorithm again to find the next Principle component. This may be applied as many times as the number of dimensions you wish to map the data to.

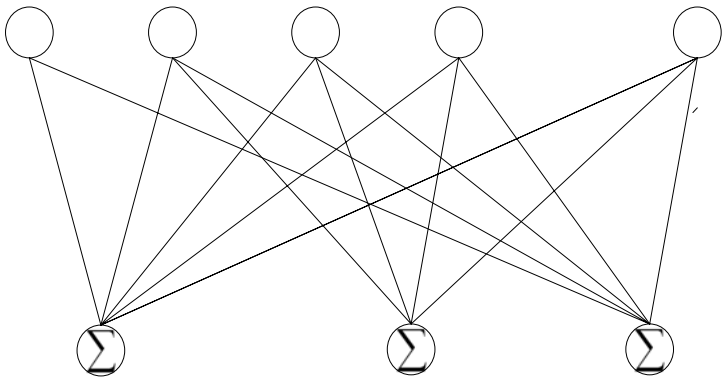


fig 6.7

Fig 6.1 shows a PCA network which will map the input vector X onto the output vector Y. Y will correspond to the first n Principle components. To achieve this mapping the following update rule is applied (Sangers rule);

D dimensional spac

$$\Delta wij(n) = \eta yi(n) \left[xj - \sum_{k=1}^{i} wkj(n)yk(n) \right] \quad \textit{(eq 6.2)}$$

$$wij(n) = wij(n) + \Delta wij(n) \quad \textit{(eq 6.3)}$$

M dimensional sp

6.3.3 K-Nearest Neighbours algorithm

This is a very naïve algorithm designed to find the most likely classification of a new point in the M-dimensional space. Using the Euclidean distance between the new point and the K-Nearest points around it, the classification is made based upon the class which a majority of the neighbours belong to.

The algorithm goes as follows;

```
algorithm K-NearestNeigbours(HTTPRequest new)

Let Q be a priority queue which holds K HTTP requests

For each trainer ∈ HTTPRequest in training set do
       Value ← Euclidean distance (new, trainer)
       Q.add(new, value)
End for

Return the most prominent classification in Q.
```

Fig 6.3.1

The Euclidean distance is calculated via the following equation;

$$E_i = \sqrt{\sum_{i=0}^{i=\text{length(n and t)}} (t_i - n_i)^2}$$

Where t_i and n_i is the value of an axis in the M dimensional space, where t is the set of training points and n is the new point.

6.3.4 Application of PCA Networks to the problem domain

PCA networks and indeed PCA in general is a very good technique for finding the parts of the input data which are most useful. The data sets that we are looking at in this project may have defining feature which accounts most of the variance in the data. This would allow us to classify the data according the score which is given at each Principle component.

The frequency of characters in the URL strings is quite likely to occur in groups, such as vowels or number characters. If this is the case it may be possible to simply extract the necessary character groups from the Principle component weights and decide the classification of the URL string using these criteria.

Another use of this technique is simply to use it for feature extraction. It may be the case that an unsupervised layer before an MLP or similar network may give better classification of the data. PCA should map the original 266 dimensional input space of character frequencies onto 10 Principle components representing maximum dispersion in the data.

6.4 Recurrent Neural Networks

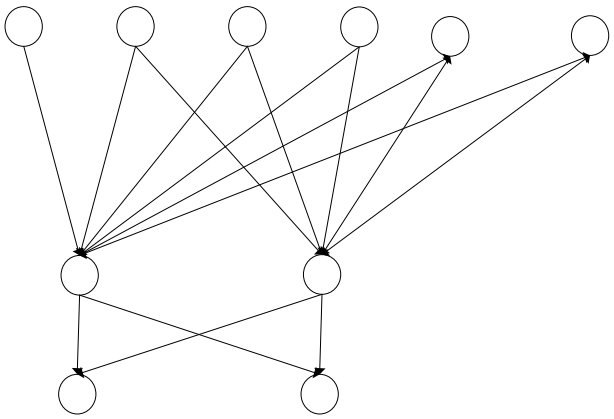
6.4.1 Discussion of Recurrent Neural Networks

Recurrent Neural Networks are Neural Network architectures which contain links both forward through the Network and backward. These architectures are therefore capable of storing some sort of memory. Of course there is much Neural Network architecture capable of representing memory in this way. The most common of these are set architectures, which are based on feed forward architectures and contain very precise recurrent links namely William Zipster [nn2] and Elman Networks [nn2, nn1]. Many people have also experimented with evolving the network architecture from scratch, although this technique gives results which are difficult to interpret.

Recurrent Neural Nets of the Elman and Zipster variety are good at tasks such as time series prediction and signal processing [nn2]. The memory which Recurrent Neural Networks are capable of storing is an important aspect in time series prediction. It allows us to find patterns which exist not just in this time instance but throughout time, in essence "unfolding" time. Allowing the Network to predict and classify structures in time [nn1].

6.4.2 Elman Networks

Jeffrey Elman (1990) proposed a simple recurrent Neural Net architecture which he named the Elman network its architecture is as follows;



The vector X represents the input vector, H the hidden layer and O the output layer. The design is very similar to an MLP except it contains an extra recurrent layer which is called the context layer and is represented by the vector C. Every time the network is activated the internal values at the hidden layer are copied up to the context layer. The context layer feeds forward in the same way as the input vector.

This allows the network to store its internal state at t-1 time steps, where t is the number of time steps, thus allowing the Network a limited amount of memory.

The weights which connect the context layer to the hidden layer can now be trained with techniques found for MLP Networks, namely Back Propagation. The Elman network trained with Back Propagation is aptly named "Back Propagation through time"; it is capable of finding patterns in time [nn1].

6.4.3 Application of Recurrent Networks to the problem domain

Of course most attackers will go through a sequence of steps whilst attempting to compromise a host. These may be performing scans of the host system for known vulnerabilities followed by an attempted exploit. These patterns an attacker goes through may be found to differ from say a normal user who continually clicks on links to legitimately navigate through a web site.

However the use of these kinds of networks becomes rather complex when we start looking at the implementation of them in an Intrusion Detection System. If we are looking at every HTTP request going through to the web server regardless of who is making the request, the patterns may be very difficult for any kind of system to spot. The other approach would be to "spawn" a Neural Network to look at each individual user's requests. This approach then complicates the system immensely; we encounter problems such as determining where the request originated from, if a bunch of people are behind Proxy connections, all users of the system will be analysed by the system as though they were a single user. Further threats to the system itself exist as people may emulate this behaviour.

Recurrent Neural Networks find structure in time, what if the structure of normal usage, or indeed misuse of the system contains no such structure. Then the system will fail.

7 Implementation and Testing

7.1.1 Commentary

The implementation of the framework was quite straight forward and seemed to work well. The multiple EvReader classes helped a lot when trying to pull data from a number of places into the same training environment. The tests which are about to be shown are those taken from four separate web sites, the first is a static, content only web site, the others are more complex sites which contain database access personal login pages and other dynamically generated content designed to test the robustness of the system. The sites have not been named here but are all existing sites which could benefit from a system such as this.

The attack data was collected from a number of sources, namely securityfocus.com, several underground hacking sites which can't be named, packet traces from the DEFCON "capture the flag" (CCTF) competition and a trace generated from the Nessus security scanner. All the traces used in these tests are approximately 5000 requests long, the data contained ranges from several distinct sessions.

7.1.2 MLP Results

The results using Multi-Layer Perceptron Neural Networks are promising. Each of the sites was trained against the collection of attacks which were accumulated. We find the results do differ between the different sites but all in all it works pretty well. The training phase consisted of reading all the requests into EvTrSet objects, these then contained marked EvHTTPRequest objects (marked as attack or not attack). This set was then split into two, picking the test and training set at random in a uniform manner. The analyser was then trained on the training data, after every 1000 epochs the results were recorded. This is what forms the basis for the following discussion. All the data presented is the unseen data, the analyser is tested with a different data set from that which it was trained.

7.1.2.1 SITE A – CONTAINS ONLY STATIC PAGES

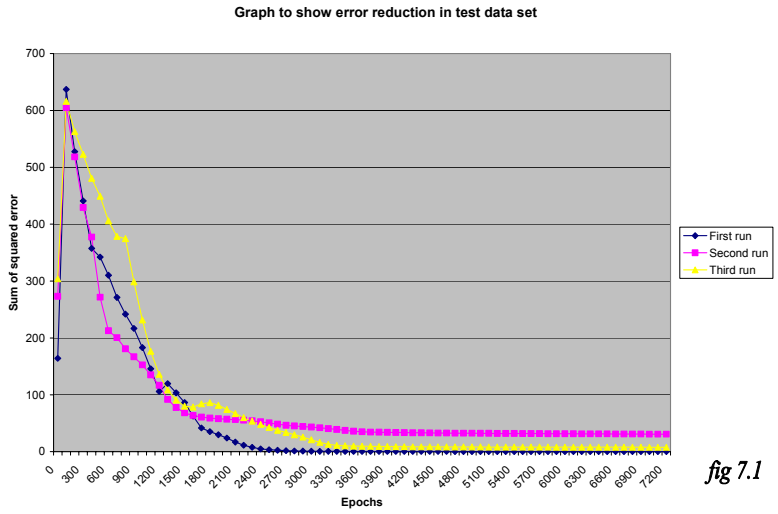


fig 7.1

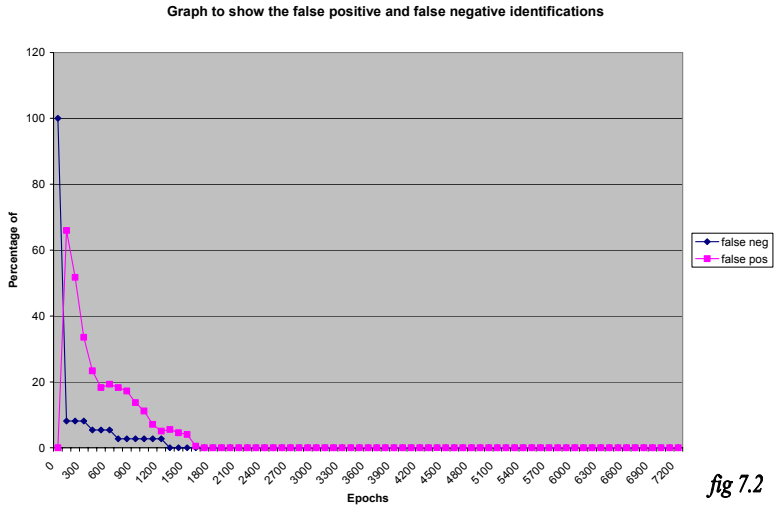


fig 7.2

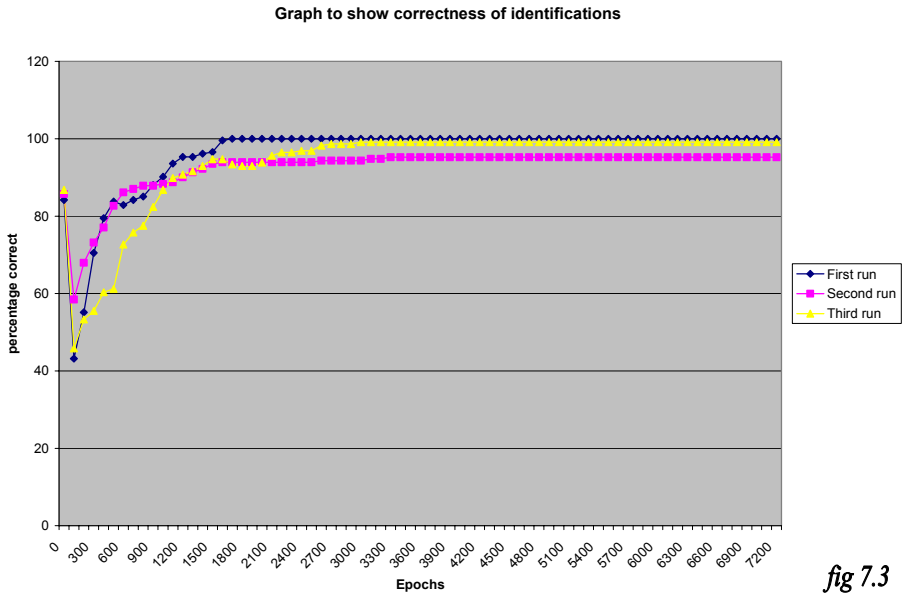


fig 7.3

	Worst	Best	Average
Correctness	95.2	100	98.11901
False Negatives	0	0	13.33333
False Positives	33.33333	0	0

fig 7.4

Fig 7.1 shows the reduction in error over the 7200 generations. We can see that each run is fairly consistent reaching similar values each time. However it is clear that the randomly chosen training and testing sets makes a different to the final result, looking at fig 7.4 we see the correctness can range from 95% right the way up to 100%. With a static website is completely feasible to make sure all the pages in the site are used in training and this will eliminate any such error.

If we take a look at fig 7.2 we see the plot of false negatives against false positives (see glossary). We notice the number of false positives start off less than the false negatives and cross at about epoch 300. This closely tallies with fig 7.1, as the error begins to fall. These values never cross again but converge closer and closer together nearing zero. On one occasion here the false positive and false negative rate both in-fact reach zero. We should notice the rather large range in false negative values here however, in the worst case we receive a false negative rate of 33%, this is clearly unacceptable and we can only speculate that it was due to a badly chosen training set.

7.1.2.2 SITE B – CONTAINS DYNAMIC PAGES

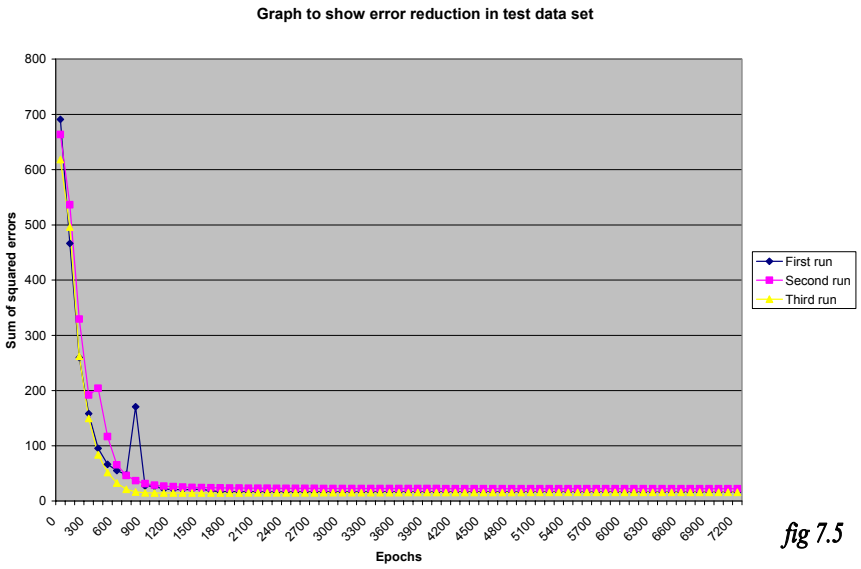


fig 7.5

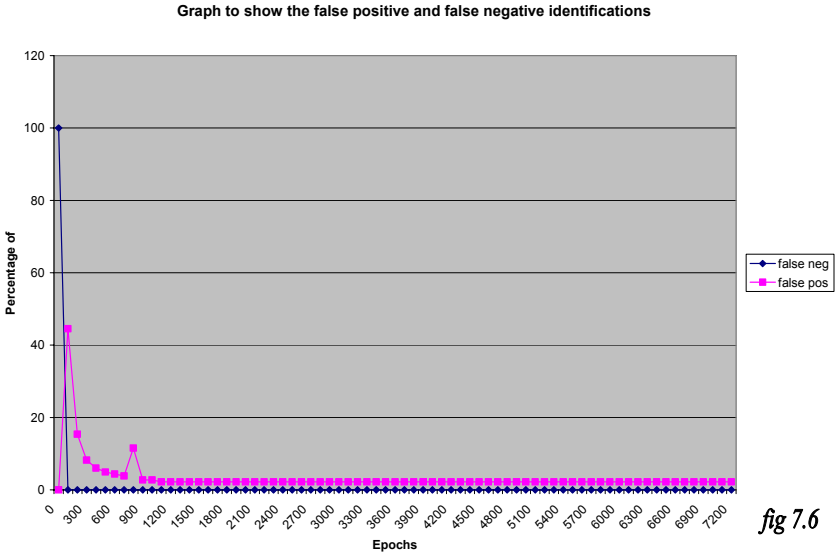


fig 7.6

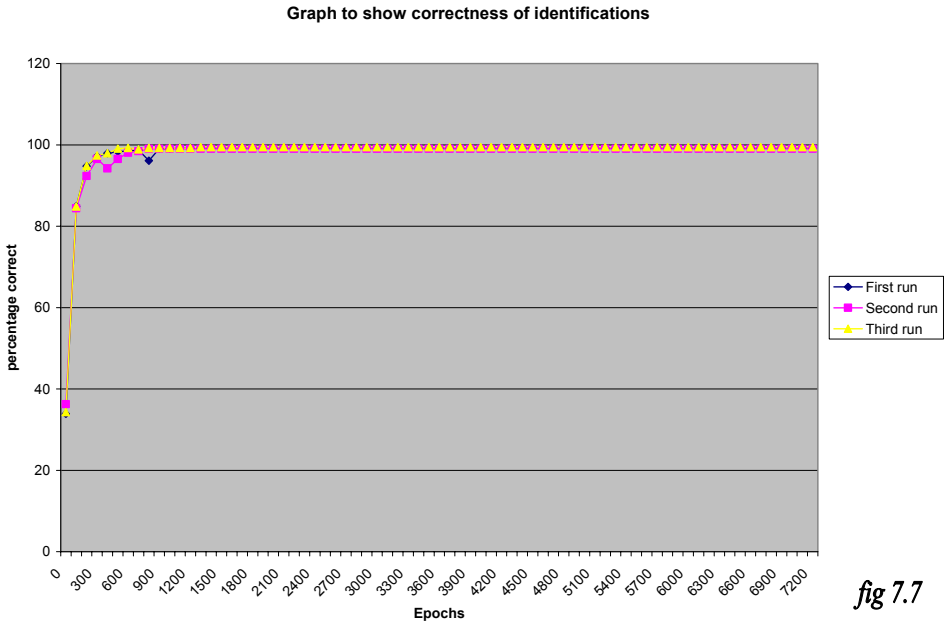


fig 7.7

	Worst	Best	Average
Correctness	99.04	99.4	99.24731
False Negatives	2.60	0.537	0
False Positives	0	0	0

fig 7.8

This web site contains log-in pages, many forms for posting data and much database access the variation in these traces is quite high. However as you can see from fig 7.5, the response to the training process is extremely promising, on each of the three runs the error fell very low. It also shows a high degree of consistency considering the random nature by which the training sets were collected. Fig 7.6 shows a similar scenario to fig 7.2 where the false negative and positive rates cross, and then converge once again. The false negative rate again reaches an outstandingly consistent value of 0%. The false negative rates could be improved upon, it is more important to allow legitimate data through so this seems reasonable.

7.1.2.3 SITE C – CONTAINS DYNAMIC PAGES

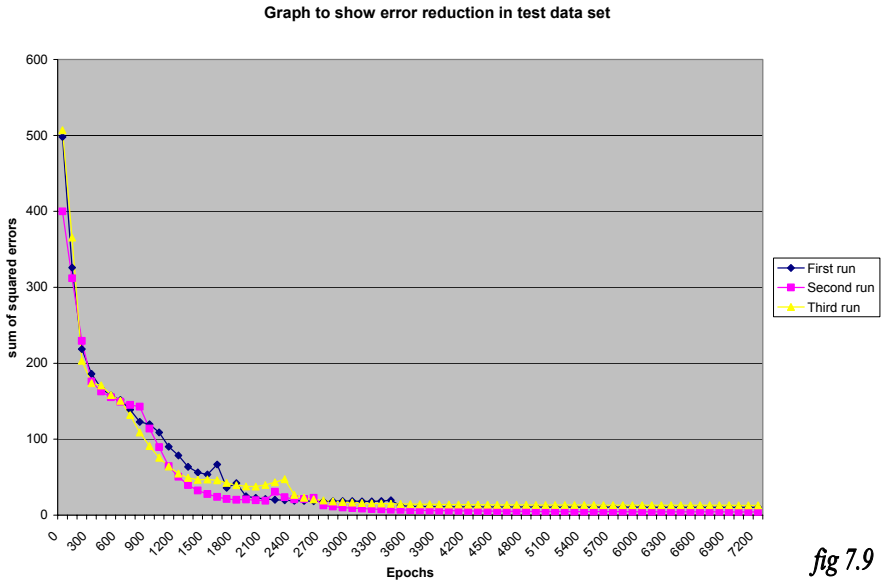


fig 7.9

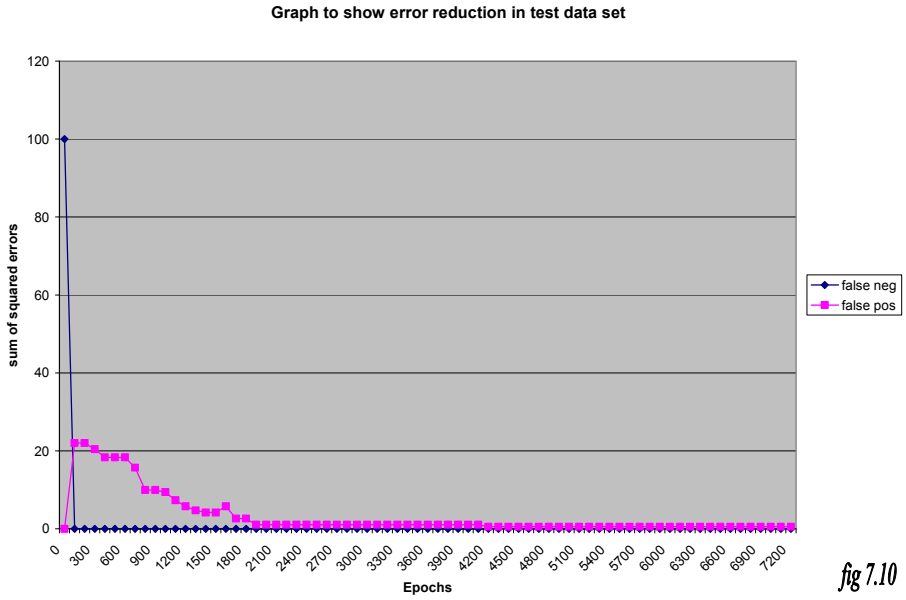


fig 7.10

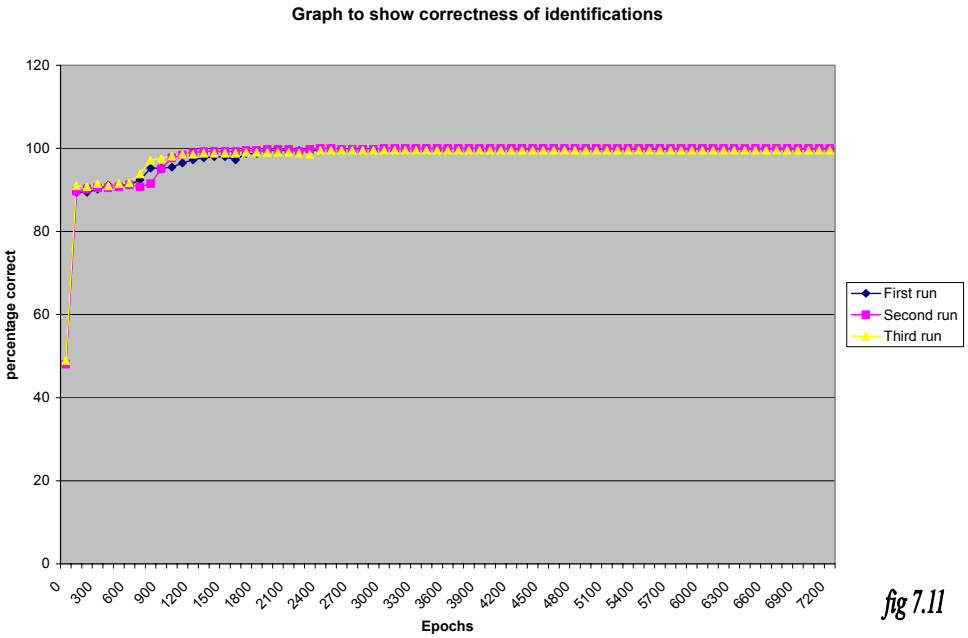


fig 7.11

	Worst	Best	Average
Correctness	99.5	100	99.75843
False Negatives	2.6	0	0.496581
False Positives	0	0	0

fig 7.12

This site took a little longer for the back-propagation to home in on the correct values, but the accuracy is unflawed. The correctness is nearing 100% on all three occasions and actually reaches 100% in one case. The best figure contains a zero false negative and false positive rates as you would expect. Fig 7.10 shows the very fast convergence of these values, this site contains much E-Commerce processing and dynamic content and these highly accurate figures come un-expected.

7.1.2.4 SITE D – CONTAINS DYNAMIC DATA

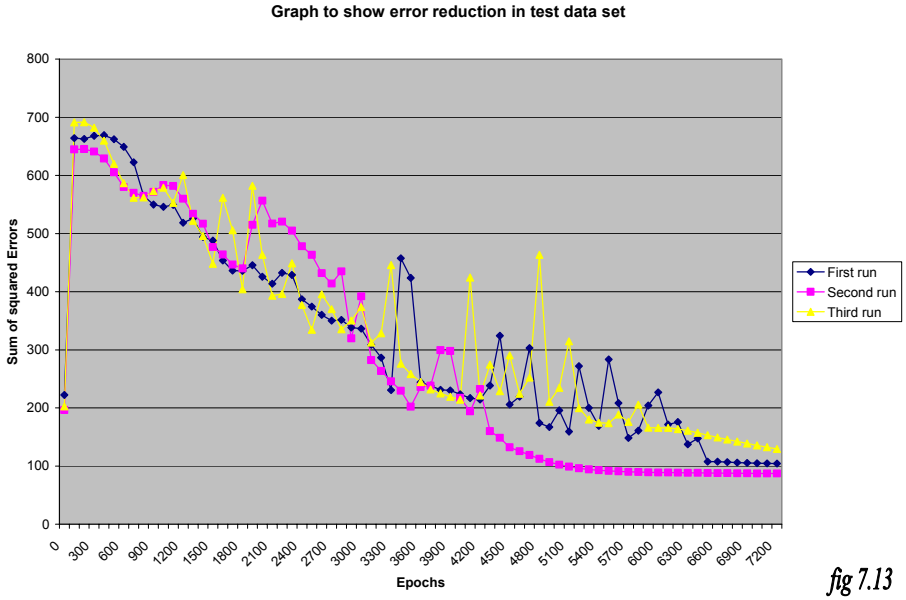


fig 7.13

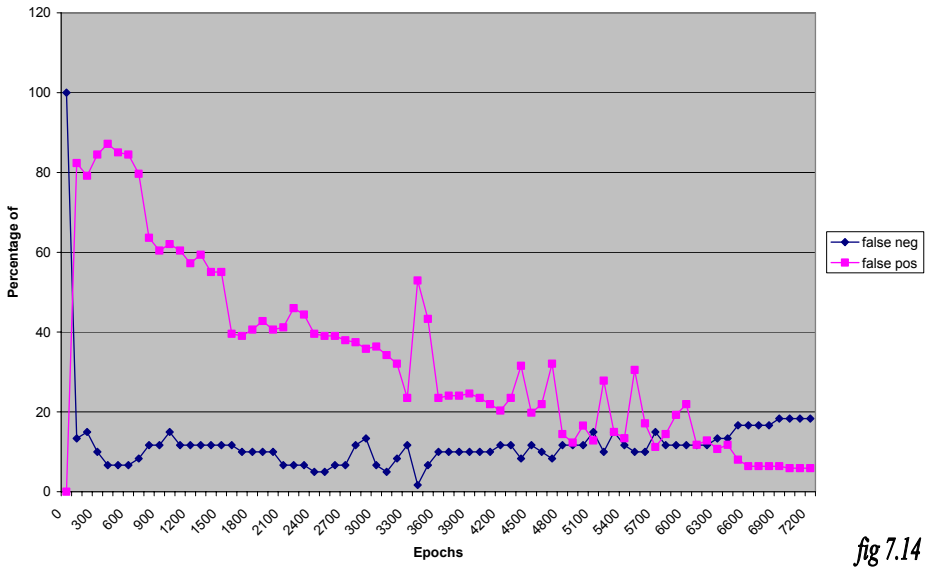


fig 7.14

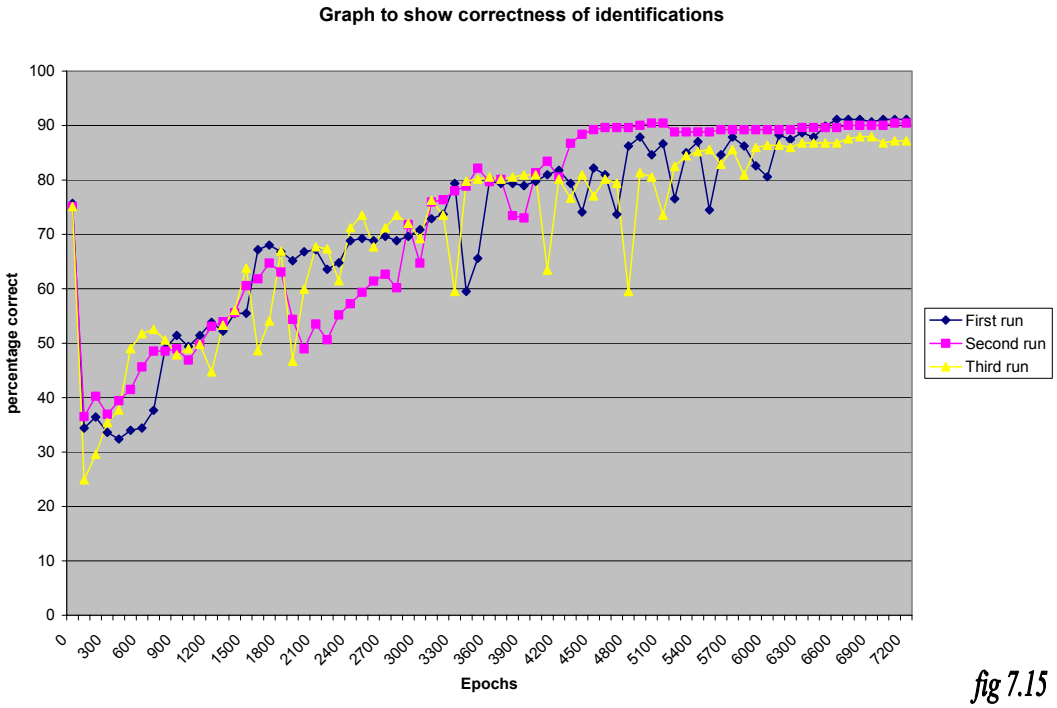


fig 7.15

	Worst	Best	Average
Correctness	87	91	89.56969
False Negatives	9.8	3.8	6.531439
False Positives	26.7	18	22.29167

fig 7.16

This site has thrown up some oddities. Fig 7.13 contains a much wilder training phase than all the other sites and produces much less quality error values. Fig 7.14 shows an erratic convergence of false positive and negative rates, however they do seem converge at one point and then diverge again. The point at which they converge (about 6300) epochs would be the point where the network is becoming over-trained and in a live environment we would stop the learning there. The final graph; shown in fig 7.15 shows an equally erratic clime to a reasonable correctness level with an average of 89%. This site doesn't seem to fit as well to the model as the others seem to. This does go to show the inherent inconsistencies between the different web sites and URL's they will produce.

7.1.3 GA versus Back Propagation

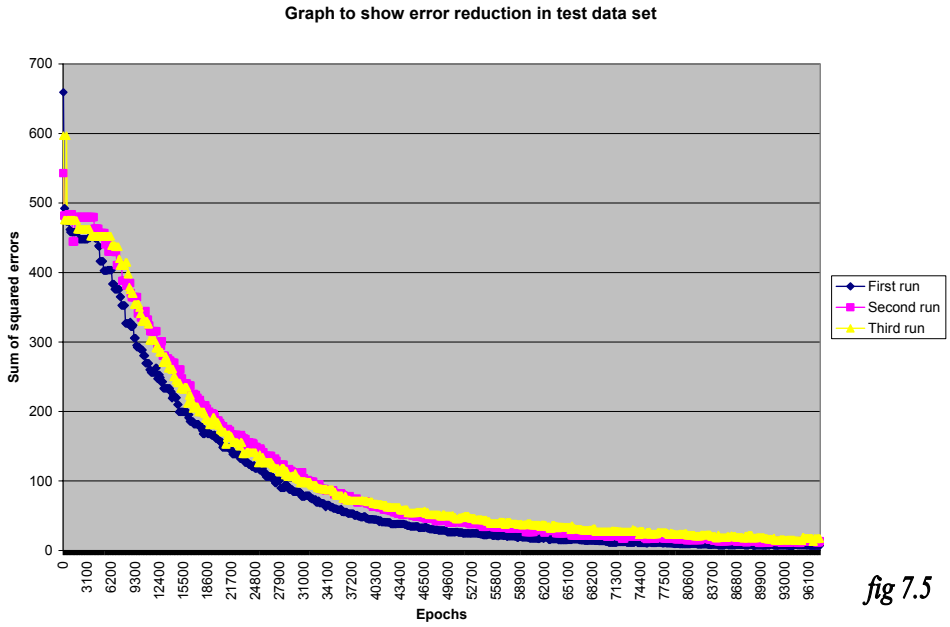


fig 7.5

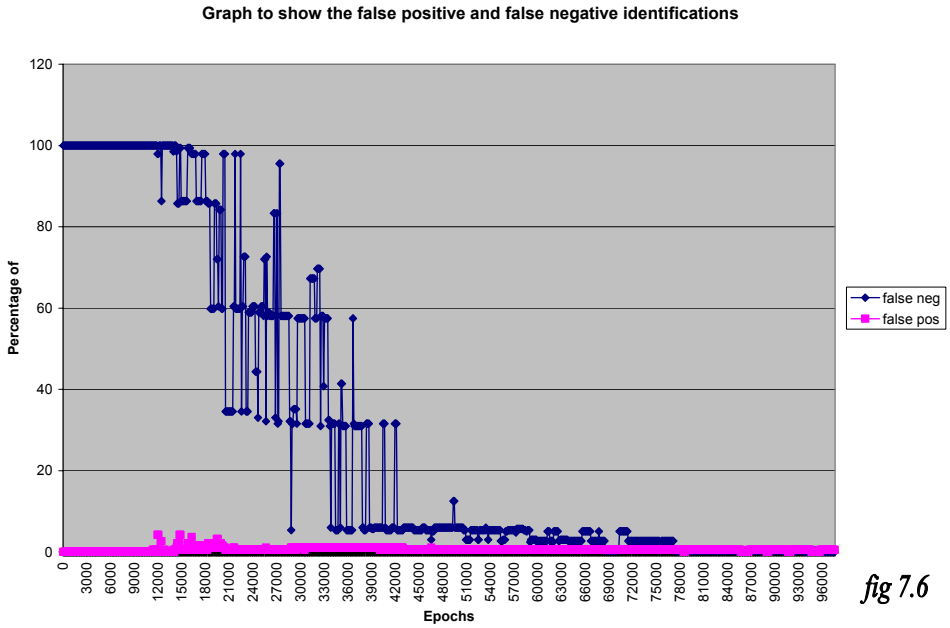


fig 7.6

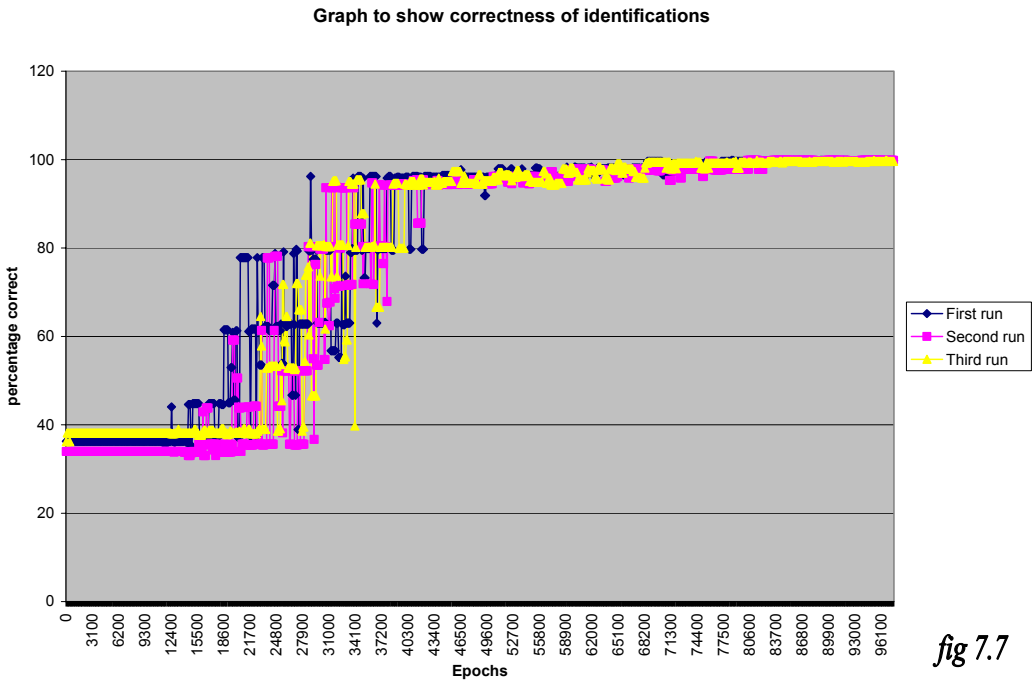


fig 7.7

We can see from these results that the Genetic Algorithm used for training does indeed find a suitable level of accuracy. However the time it takes to do so is much greater than the Back Propagation algorithm and so its use is no longer considered. The nature of the Genetic Algorithm means that the weights will be manipulated in a much more random fashion and provide much less precise guidance into a reasonable error value.

7.1.4 PCA using unsupervised Neural Nets Results

The PCA results look a little unnerving. They reveal a very complex search space which seems to be much more complex than first thought.

7.1.4.1 PCA RESULTS PLOTTED IN 2-D SPACE

Site A

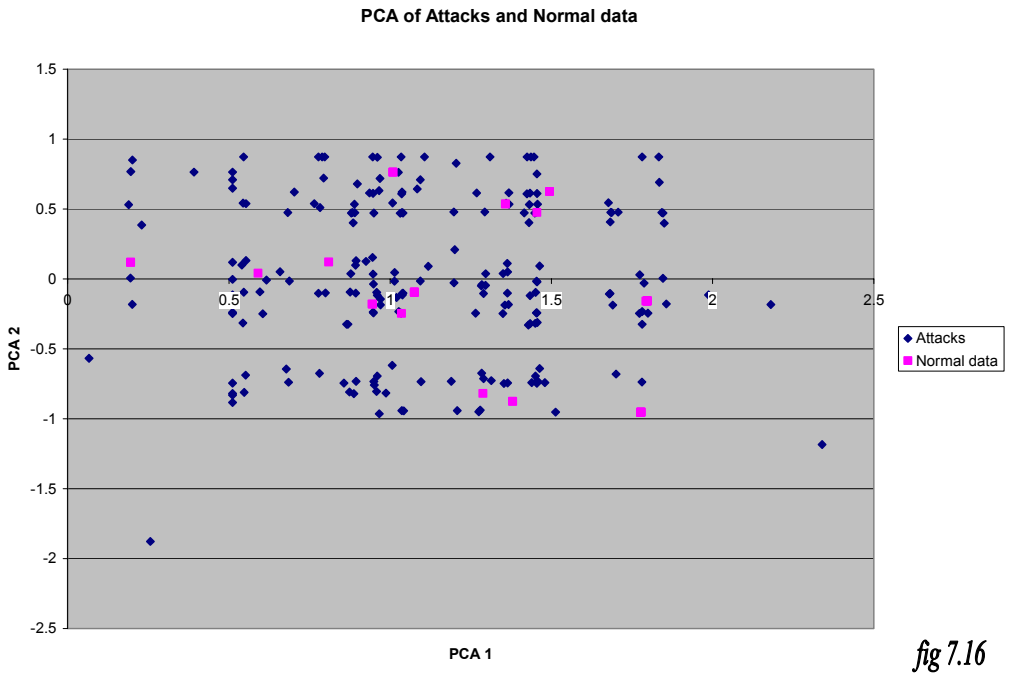


fig 7.16

Site B

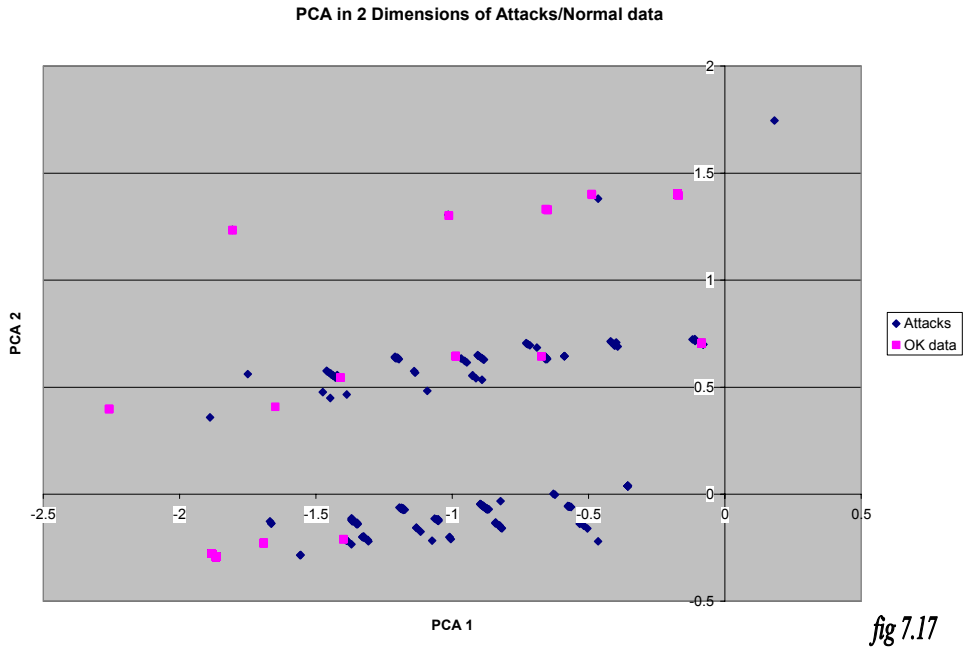


fig 7.17

Site C

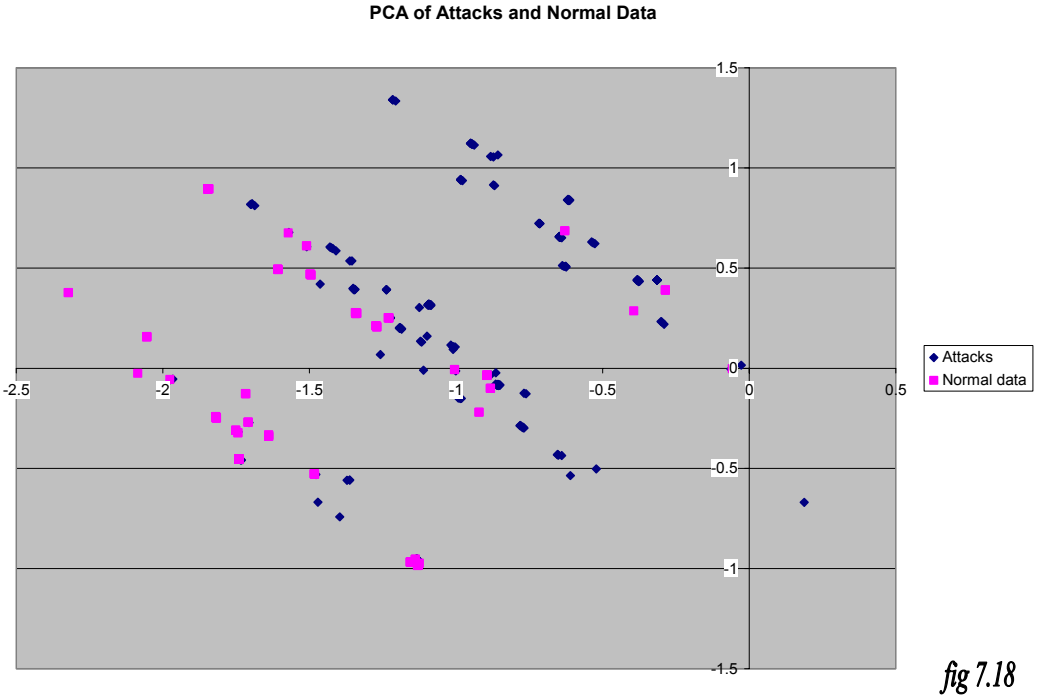


fig 7.18

Site D

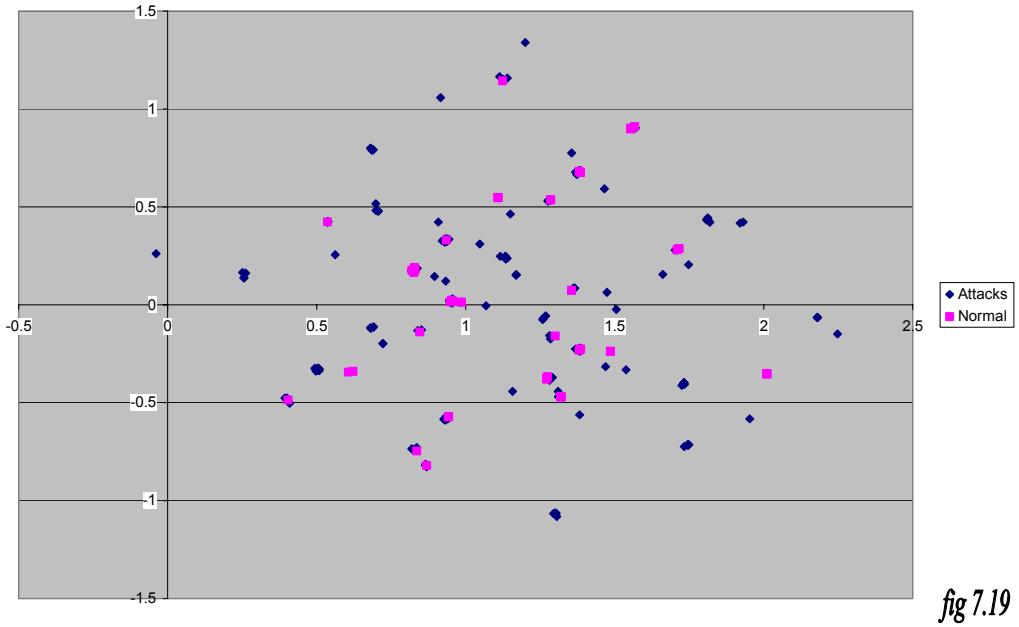


fig 7.19

All Sites

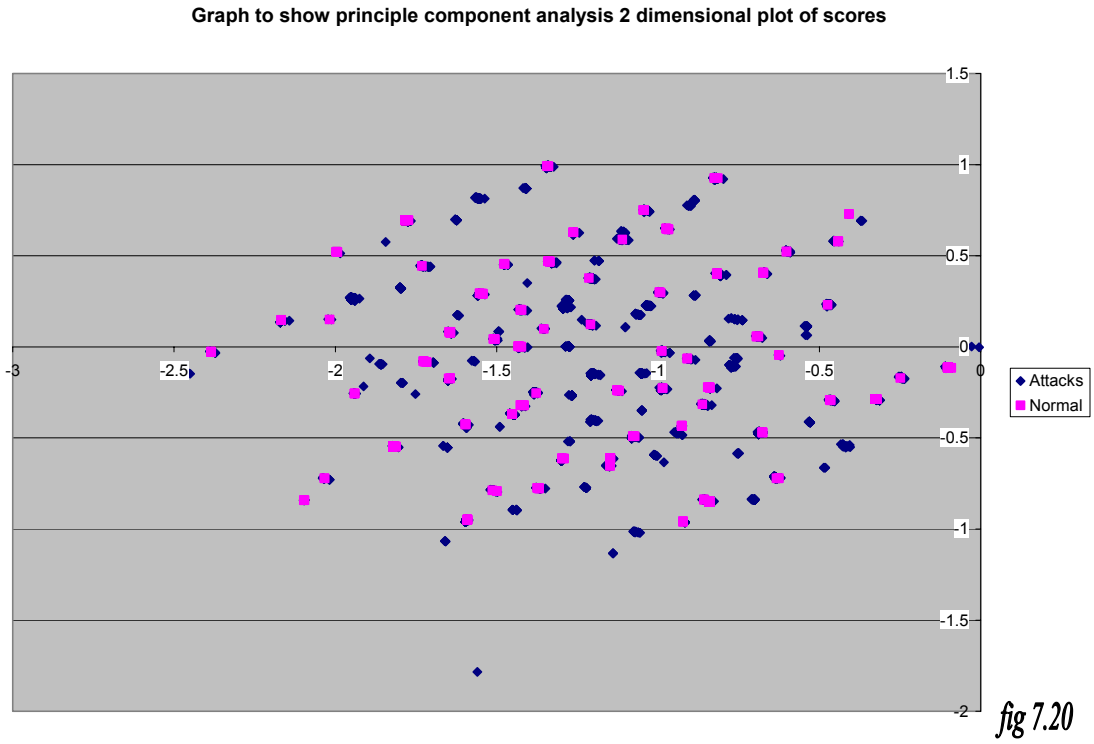


fig 7.20

These results look a bit messy. There seems to be no clear split between the two classes for any of the sites. Site C seems to give the most clear cut distinction. It may be necessary to look at a higher dimensional space, in essence finding more Principle Components. Of course these cannot be plotted in a graph as the dimensionality is too high.

7.1.4.2 PCA RESULTS AS BAR CHART OF SCORES OF FIRST TEN PRINCIPLE COMPONENTS

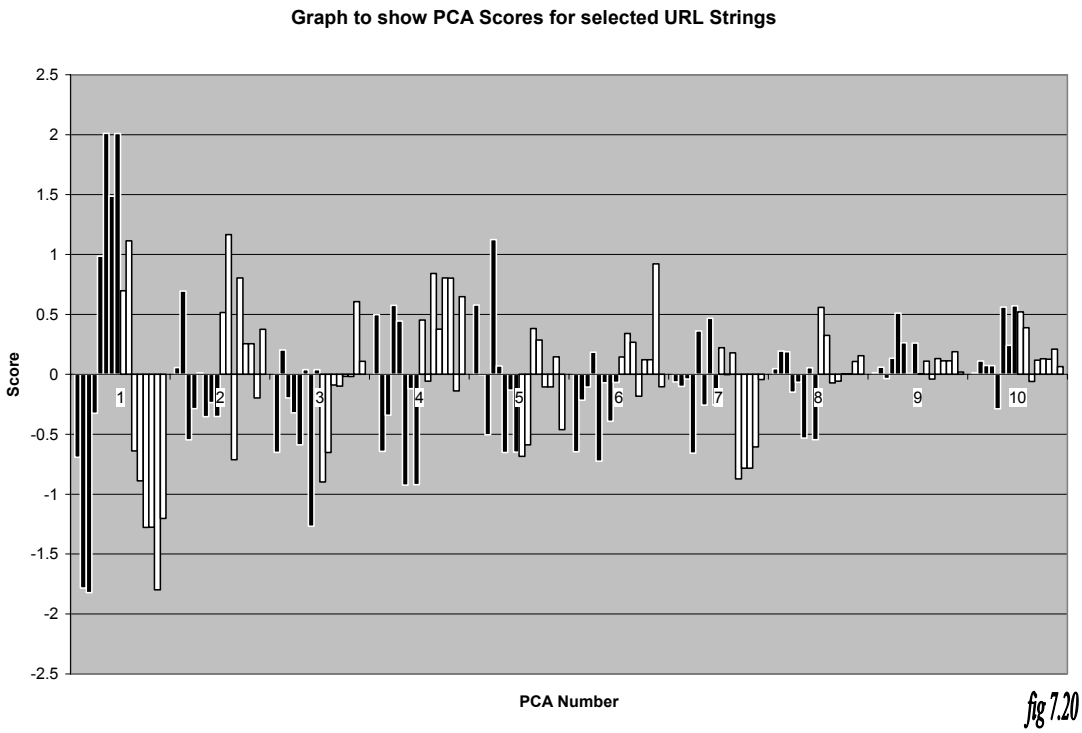


fig 7.20

….. Normal Data

….. Attack Data

Fig 7.20 shows the results of PCA for the first ten Principle Components. We see no real correlation between any of the values here. When we look at what the actual Principle Components are however we do see some interesting things appearing.

Site A

9 98342157][SIZ

PCA 1:
[/ 0.1772155372581642][SIZE]
PCA 2:
[SIZE]
PCA 3:
[/ 0.11229033918366738][SIZE]
PCA 4:
[/ -0.06042977174635738][SIZE]
PCA 5:
[/ 0.07805356325227099][SIZE]
PCA 6:
[/ -0.8885027417410923][X 0.24137106922559015][?
[SIZE]
PCA 7:
[/ 0.3652904215371211][X 0.513723365702686][i -0.05
PCA 8:
[. 0.1496524037795909][X -0.20608417999255246][SIZE]
PCA 9:
[- 0.09350104398535186][. -0.0920305085532854][X -0
0.07898692783534186][c 0.253993136710603][g 0.233177
0.3286077707633603][n 0.10699387533687275][o 0.05507

All the sites seem to pick out the '/' and '.' characters a lot, this is used in directory traversal attacks so this does has some significance. The other thing to notice is the use the character 'X', it happens that most of the buffer overflow attacks use this character for padding the buffer giving this relevancy. Also notice that the size of the URL is quite probably the most significant thing in each of the sites. We also notice vowels such as 'i' and 'e' popping up occasionally, of course most standard web site URLs' are likely to contain vowels. This PCA appears to give us some information in the analysis of what the MLP Neural Net may be exploiting, but seems unlikely to aid us in the classification task, as the scatter graphs of section 7.1.3.1 show.

7.1.4.3 PCA RESULTS – APPLICATION OF 10 NEARESET NEIGHBOURS IN 10-D SPACE

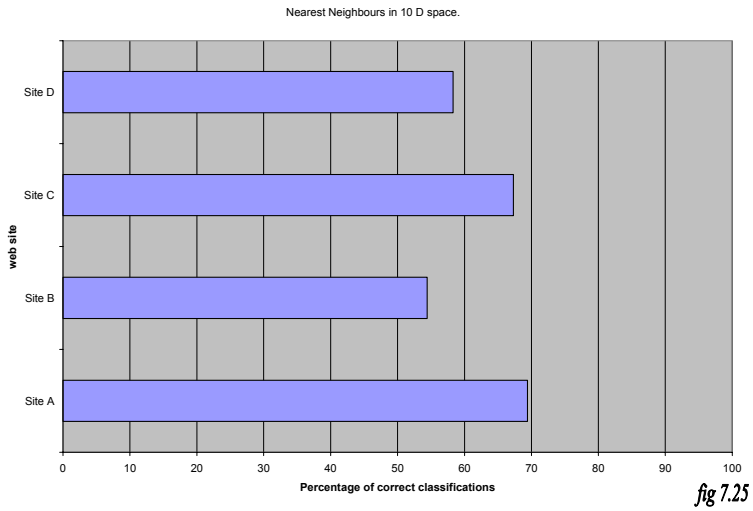


fig 7.25

The Nearest Neighbours algorithm applied to the 10 dimensional space plotted from the scores of the PCA gives reasonable accuracies. We see ranges from mid fifties to seventy percent accuracy; this shows the application of PCA is mapping the output onto a reasonable lower dimensional space which allows for some crude classification. Of course an MLP Neural Network may be able to use this pre-processed information and increase the accuracy of the classifications but it is promising to know that the points of interest are fairly close together after PCA in 10 dimensions.

7.1.4.4 PCA RESULTS AS PRE-PROCESSOR FOR AN MLP NEURAL NETWORK

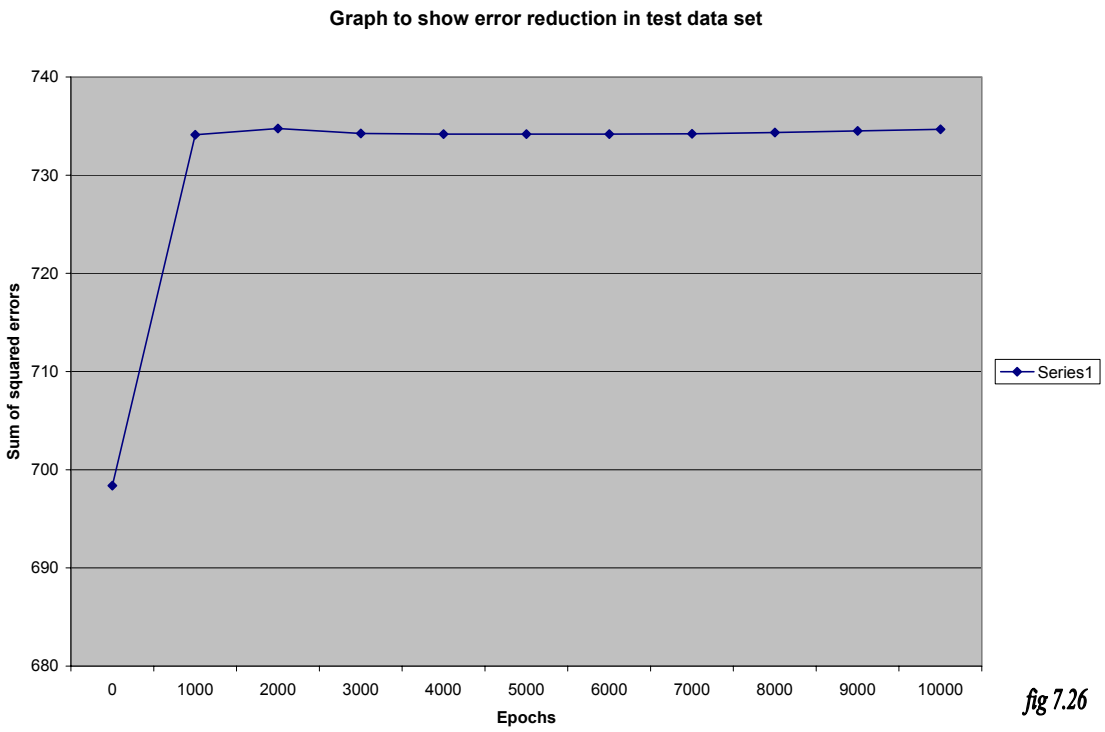


fig 7.26

The PCA network did not provide as good a solution as expected! The graph in fig 7.26 shows an example of an appalling run. The elapsed time for this to run was just over 8 hours! It provides no indication of convergence to a reasonable error value. This shows that PCA does not provide a very good pre-processor for the MLP for this problem. PCA in general has been useful in some instances and not in others a further investigation into unsupervised learning and other statistical techniques is a worthy topic of investigation in the future.

7.1.5 Letter group occurrence analysis

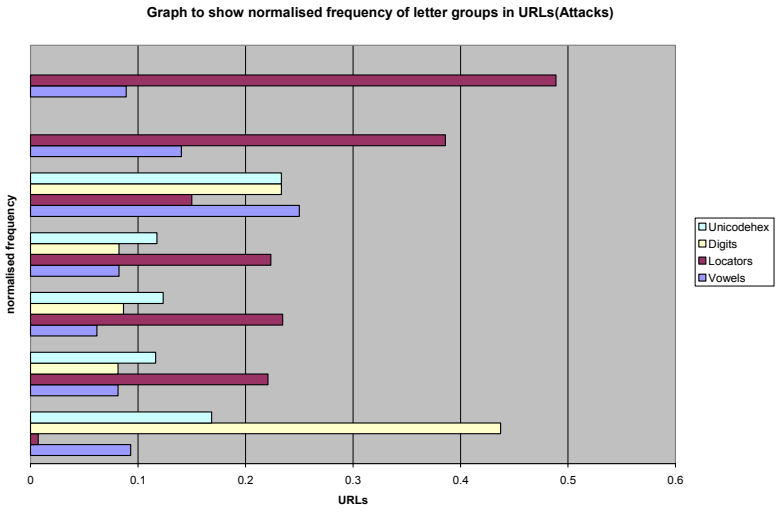


fig 7.27

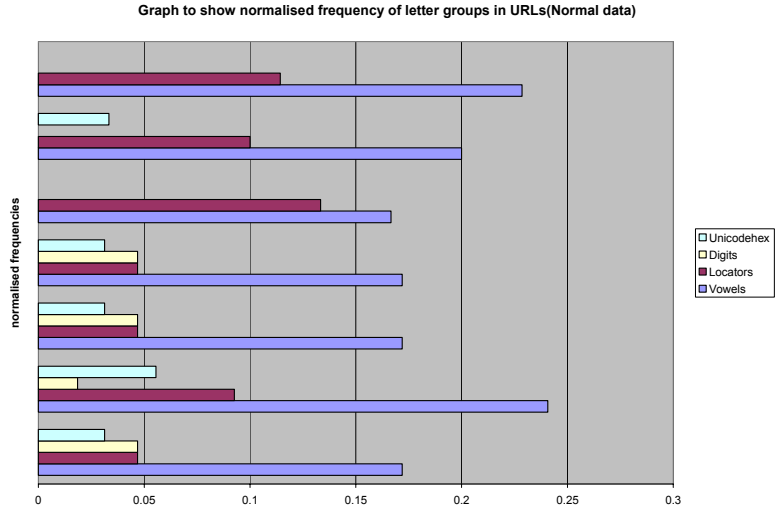


fig 7.28

From fig 7.27 and fig 7.28 we can generally see a lower frequency of vowels in the attack strings. This is probably due to the nature of an attack, as other characters such as buffer overflow padding an egg code is normally added, this will wash out the vowel frequency. We also notice the increased use of Unicode and hexadecimal values; these are not normally attributed to normal URL strings to the same extent.

7.1.6 Recurrent Neural Net Results

An Elman Network (section 6.4.3) was implemented in the original version of this project (Appendix 10.1.2). The use of Recurrent Neural Nets (RNNs') required a completely different approach to the design of the analysis and event engines. The use of sessions, whereby a Network would look after a particular user's session and analyse its behaviour was used. The use of such a complex framework meant that the complexity of the program escalated.

Many problems emerged, such as how to decide which session a particular request belonged to. The HTTP protocol does not contain the concept of sessions in this sense (of course Session state can be used). Other problems emerged such as the spoofing of IP Addresses, which will throw off any session based analysis. Other issues include the use of Proxy Servers, if the user of the web site is behind a Proxy Server the users from behind the proxy will look like they are from the same IP address. This complicates matters further and leads to a very unstable design which has no practical use.

The actual use of RNNs' becomes very complex when dealing with patterns over time. Ignoring all the problems stated above, the system was implemented. Sessions were hand crafted to ensure that they were correct and the data sets were created. The Neural Network was able to learn the patterns of session based activity, but when given a new session it had not seen before, the system failed spectacularly. It seems that Elman networks are very capable of learning the patterns of requests through time but have absolutely no chance of any generalization task. The search space appeared very complex and no real patterns were present in requests in time.

7.1.7 Deployment of a live system

7.1.7.1 COMMENTARY

The system can be deployed in a number of ways. The EvReader interface (section 5.2.3) can be implemented allowing a generic way for the AnalysisEngine to receive analysis requests. Several ideas were discussed and the system implements two such systems as live systems which are ready to be used.

The first idea for the deployment of the URL scanner would be to use it inside the Web server itself. This would require adding the system to a current Web server or indeed building a new one. This would mean that each request received is analysed and the connection abandoned if deemed malicious.

The second idea for deployment is to use the library libpcap which was used in training. This system provides a nice structure to access Ethernet cards and would be an implementation of a packet sniffer/sucker.

The final idea for deployment of our system is to implement an application level firewall. This would be a Proxy Server that forwards requests to the Web Server. Requests would not be forwarded if the analysis engine deemed the request malicious.

The final implementation only uses the latter two, below is a discussion of their implementations.

7.1.7.2 PACKET SNIFFER

The Packet Sniffer is implemented in the class EvReaderJPCap. Its use implements the EvReader interface using the JPCap library. This library is a java wrapper for libpcap, the infamous UNIX packet capture library. Implementation was fairly straightforward as the JPCap library offers a lot of functionality which saves time.

The problem with this kind of system is it is really a detection system. The system will run on a computer on the same network as the Web Server, or indeed the same machine as the Web Server. If the system is on different machine the network card of that system will need to be set to promiscuous (see glossary). Also this topology will not work unless the hub is not switched. If it is a switched hub or router the requests will not reach the Packet Sniffer.

If the Packet Sniffer sees a packet it thinks is malicious it will log the event. A statistics broadcast is displayed every few minutes to inform people of the number of attempted attacks.

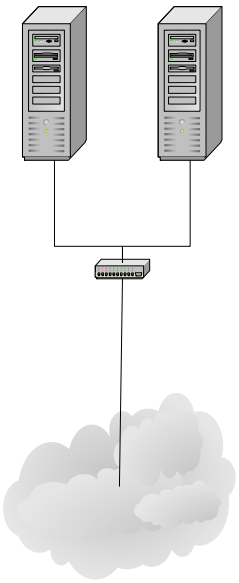
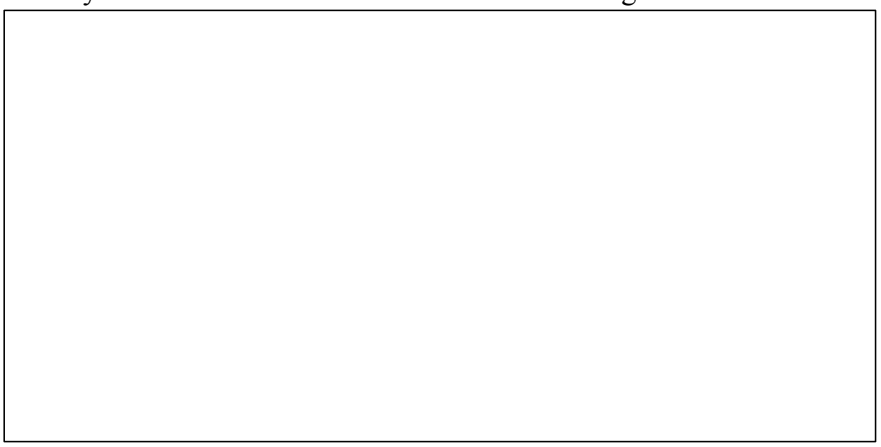


Fig 7.29 shows a typical installation of such a system. Any attempt to attack the Web Server will be recorded for analysis by the Packet Sniffer. It is better to have it on another machine, because if the Web Server is compromised the log files could be erased.

Fig 7.30 shows a console output of this system running. Appendix 10.3.1 shows the results of analysis of some selected data in the form of a log file.

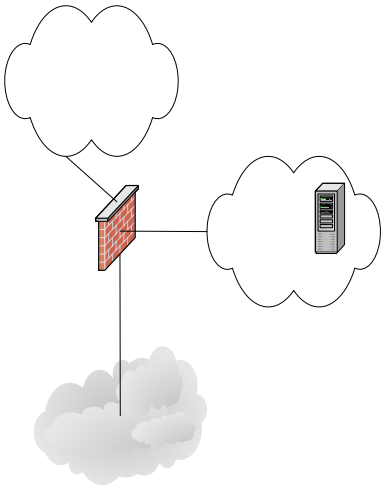


7.1.7.3 APPLICATION LEVEL FIREWALL, PROXY SERVER

A firewall is normally associated with a system which blocks particular IP, TCP, UDP, ICMP etc packets according to a set of rules. IPCHAINS and Microsoft's ISA are such systems. Firewalls can however be at the application level applying rules for the passing of packets. It seems like a good idea to include an application firewall approach to the deployment of the system. Reasons for this include the benefits of a standard firewall;

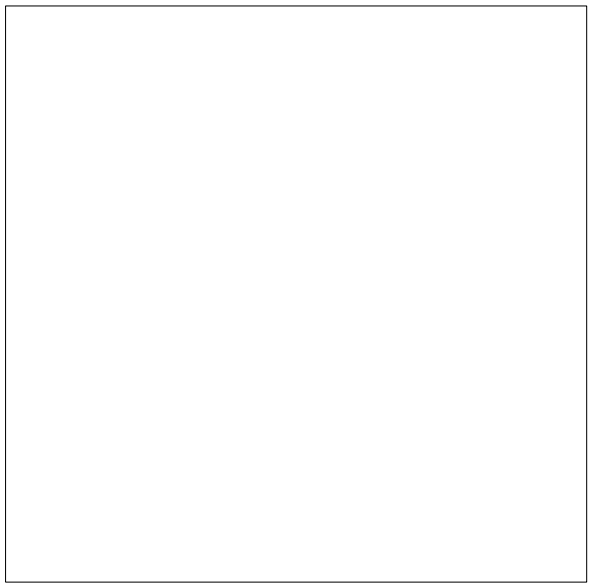
- Requests that do not meet the criteria specified are simply not allowed to pass, so the success of the attack is greatly reduced.
- The Web Server is not less likely to receive garbage which will eat resources on the system.
- The implementation is Web Server independent, i.e. it will work on IIS and Apache.

Below is the architecture of a typical implementation of such a system;



The simplest way to implement this kind of system is to write a Proxy Server. This is a server which sits on the firewall computer. Requests for the Web Server are first dealt with by our Proxy Server. If the request passes analysis it can be forwarded on to the Web Server and the response given to the client. It is important to give a confusing message to the attacker who is attempting the attack. Such a message would be to close the connection, or to return a page not found error or such like.

Here is the console output when running this system when an attack is received;



8 Conclusion

The goal of the system was to investigate the field of Intrusion Detection in an attempt to apply Artificial Neural Networks to the problem domain. Contrary to most other Intrusion Detection Systems we looked not at the transport and network layers but at the application level protocols, namely the HTTP protocol.

A number of techniques were applied in analyzing HTTP request data, in an attempt to detect attacks whilst keeping the false alarm rate to a minimum. [ids12] tells us that the false alarm rate is actually the limiting factor and shows the system to be useful. System administrators will become much more willing to investigate an attack attempt if the false alarm rate is low, or in fact non-existent. The results found in this investigation show varying degrees of success in this task.

With an MLP trained with Back Propagation, static web sites seem to be able to gain an accuracy of 100%. Dynamic web sites provide less accurate results but these results are still in the high 90's. We should notice that two out of the three dynamic web pages were able to gain false alarm rates (false positives) at 0% on more than one occasion. These results are very promising. Further investigation could be done into discovering what sorts of URL strings help the system learn better and guidelines for users of the system could be produced.

When we replaced the Back Propagation algorithm with a Genetic Algorithm (GA) we found that the GA was capable of finding a suitable set of weights. However the length of time the Back Propagation algorithm took to train was far shorter than the GA. The superiority of the Back Propagation algorithm led us to no longer consider the GA in the testing process.

PCA provided little information as to the workings of the system. It did provide us with some insight into the kinds of characters we see in the normal and attack data. This information could have been useful in constructing a pre-processor for our MLP Neural Net but actually provided a very poor solution.

Recurrent Neural Nets were looked into as an analytical technique. This task became quite a major one and the investigation into the Elman network was aborted due to the wildly poor results.

Finally we looked at a very simple analysis of groups of letters in order to discover if there was some simple way to classify the data. What we found was some correlation between the characters involved in attacks and those not, but a significantly obvious relationship did not exist.

The other functional requirements were fulfilled as follows;

1. The system must be able to read HTTP messages from a variety of sources. These must include raw network packets, log files and TCP/IP sockets.

This requirement was met through the EvReader interface and its implementation see section 5.2.3.

2. The user will be able collect the necessary training data using an appropriate tool.

This requirement was fulfilled, with the use of the EvReader implementation using the packet capture library JPCap. The EvReaderJPCap reader can be combined with the AeAnalyserDump class which dumps the data to the disk for further analysis later.

3. The user will be able to train the neural network based analysis engine.

The WiTrainEnv and the WiComponentFactory object allow the user to configure the training environment in which the system is trained. The highly configurable nature of the system allows the user to implement their own AeAnalyser objects if they wish.

4. The system must be able to be started and stopped by a privileged user of the system.

The system requires root access to the system in order to open the EvReaderJPCap in live mode. It may only then be killed by a privileged user of the system.

5. The system will be capable of learning the difference between a normal HTTP message and an anomalous one.

This was the main goal of the project. This requirement has been full-filled as the analysis engine provides high levels of accuracy.

6. *The system is required to notify a system administrator when a sequence of events that is likely to be an attack is encountered.*

The framework of the system allows a system administrator to register an EvObserver object with the Event Engine. If a possible attack is detected the Event Engine will then notify the registered observer objects. Administrators may implement these objects as they choose.

7. *Statistics of the systems operation must be able to be collected for analysis and review.*

Each AeAnalyser object writes out statistics regarding the analyzers performance.

The non-functional requirements were also fulfilled as stated;

1. *The system must be easy to train for new users of the software. The mechanism for training the system used must be easy to understand and allow someone with a*

reasonable background in computer networks to capture the correct data and use the training software to create an analysis engine for their network.

This requirement has been fulfilled as much as possible, although the system is fairly difficult to set up and use. Future work could make this part of the system nicer.

2. *Users of the system must be able to start and stop the system easily.*

This is simply a matter of being root and typing;

java WebIDS.WiTrainEnv
or java WebIDS.WiOpenLive

into the console, and so anyone with any knowledge of UNIX should be able operate the starting and stopping of the system.

3. *Users of the system must be able to configure any parameters the system may use in an easily manageable way.*

All parameters are configured from constant values in individual class files. Main things, such as which EvReader and which AeAnalyser to use are configured through the WiComponentFactory class. This seems reasonable, although it could be improved upon.

This project was investigation into Intrusion Detection within a Web Server. We found a suitable method of analysis by using an MLP Neural Network. Each requirement was fulfilled to a degree.

The project has opened up many avenues of investigation. Namely Recurrent Neural Networks and their application to this kind of system may provide a better analytical technique. Other future work includes the use different Neural Network architectures which may or may not be better suited to this kind of analysis.

Research within the IDS area tends to be heading more towards a distributed approach, where a number of analytical techniques are employed looking at several avenues within the system [ids2]. These kinds of systems can then draw correlations between a number of different analytical paths within the system.

Finally, this project was looking at the HTTP protocol. It would be interesting to see whether other internet protocols respond to this kind of analysis in a similar manner. The use of these techniques on encrypted protocols, such as SSH unearth another potential problem.

9 References

These many of these references are also discussed in the literature survey in Appendix $X.

[se0] Phrack 49 Oo Vol 7 Issue 49 Smashing the Stack for fun and profit

[se1] Mudge: How to write Buffer Overflows Tutorial

[se2] scutt/team teso, Exploiting Format String vunerabilities.

[se3] rain.forest.puppy, Perl CGI problems, Phrack Issue 55

[se4] Gregory Gilliss, CGI Security Holes, Phrack Magazine,Volume Seven, Issue Forty-Nine

[se5] rain.forest.puppy, NT Web Technology Vulnerabilities, Phrack Magazine, Volume 8, Issue 54 Dec 25th, 1998

[se6] Scambray Joel, McClure Stuart and Kurtz George: Hacking Exposed

[se7] horizon jmcdonal@unf.edu, Defeating Sniffers and Intrusion Detection Systems,Phrack Magazine Volume 8, Issue 54 Dec 25th, 1998, article 10 of 12

[se8] http://www.theregister.co.uk/content/archive/22010.html

[ids0] Helmer Guy G, Wong Johny S.K, Honavar Vasant and Miller Les. Intelligent Agents for Intrusion Detection. Ames, Iowa.

[ids1] Cannady James, Artificial Neural Networks for Misuse Detection.

[ids2]Balasubramaniyan Sundar Jai, Garcia-Fernandez Omar Jose, Isacoff David, Spafford Eugene, Zamboni Diego, An Architecture for Intrusion Detection using Autonomous Agents.

[ids3] Ghosh Anup K, Schwartzbard Aaron and Schatz Michael, Learning Program Behavior profiles for Intrusion Detection.

[ids4] Ghosh Anup K and Schwartzbard Aaron, A Study using Neural Networks for Anomaly and Misuse Detection.

[ids5] Ranum J Marcus, LangField Kent, Stolarchuk Mike, Sienkiewicz Mark, Lambeth Andrew, Wall Eric, Implementing a Generalized tool for network monitoring.

[ids6] Paxton Vern Bro: A System for Detecting Network Intruders in Real-Time

[ids7] Cannady James, Next Generation Intrusion Detection: Autonomous Reinforcement Learning of Network Attacks.

[ids8] Schuba Christoph L, Krsul Ivan V, Kuhn Marcus G, Spafford Eugene H, Sundaram Aurobindom, Diego Zamboni: Analysis of Denial of Service Attack on TCP.

[ids9] Machine Learning Techniques for the Domain of Anomaly Detection for Computer Security

[ids10] Northcutt Stephen, Cooper Mark, Fearnow Matt, Karen Fredrick. Intrusion Detection Signatures and Analysis. Chpt 10-11

[ids11] 0x0b[0x10], A STRICT ANOMOLY DETECTION MODEL FOR IDS, Volume 0xa Issue 0x 05.01.2000 - P H R A C K M A G A Z I N E –

[ids12] The Base-Rate Falacy and its implications for the difficulty of Intrusion Setection

[nn0] Bishop Christopher M: Neural Networks for pattern recognition Chpt 8.

[nn1] Jeffrey Elman, Finding Structure in Time, 1990.

[nn2] Danilo P. Mandic, Jonathon A. Chambers. Recurrent Neural Networks for prediction. Learning Algorithms, Architectures and stability chpt 5.

[nn3] Principe Jose C, Euliano Neil R, Lefebvre W Curt. Neural and Adaptive Systems Chapter 6, Chapter 3;

[nn4] Russel Stuart, Norvig Peter. Artificial Intelligence A Modern Approach.

[nn5] Ellis David. Non-Symbolic AI assignment; Training Algorithms for MLP networks.

[nn6] Recurrent Networks [1,2,3], *http://www.williamette.edu/~gorr/classes/cs449/rnn1.html* up on 22/12/01

[nn7] Haselsteiner Ernst, What Elman Networks Cannot Do.

[n0] Stevens W. Richard: TCP/IP Illustrated Volume 1, The Protocols, Volume 2 The implementation, Volume 3 TCP for transactions, HTTP, NNTP and UNIX domain protocols.

[n1] Packet Capture with libpcap and other low level network tricks.

[n2] W. Richard Stevens. TCP/IP Illustrated Volume 3. Chapters 12/13 (HTTP Protocol/Packets found on an HTTP server).

[n3] RFC HTTP/1.1.

[n4] http://www.mit.edu/people/mkgray/net/web-growth-summary.html up on 22/12/01

[r1] Gamma Erich, Helm Richard, Johnson Ralph, Vlissides John, Design Patterns 1995. Pages 293-314.

[r2] Perdita Stevens,Pooley Rob, Using UML.

10 Appendices

10.1 Source Code

10.1.1 WebIDS version 2 – Final Version

10.1.1.1 WEBIDS PACKAGE


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS

10.1.1.2 WEBIDS.EVENTENGINE PACKAGE


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS

10.1.1.3 WEBIDS.EVENTENGINE.PARSEHTTP PACKAGE


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS

10.1.1.4 WEBIDS.ANALYSISENGINE PACKAGE


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS

10.1.2 WebIDS version 1 (aka IDSSystem) – Original Version (uses session based analysis).


Web
IDS


Web
IDS


Web
IDS

10.1.3 WebIDS version 1 (aka IDSSystem) – Original Version (uses session based analysis).

10.1.3.1 IDSSYSTEM PACKAGE


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS

10.1.3.2 IDSSYSTEM.EVENTENGINE


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS

10.1.3.3 IDSSYSTEM.ANALYSISENGINE


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS


Web
IDS

10.2 Literature Survey Research Comments by D.Ellis

This literature survey is divided into three sections; the first section refers to material covering security exploits that pose threats to computer systems. These include tutorials on buffer overflows, Unicode, CGI exploits and format string exploits.

The second section covers current Intrusion Detection research; some of the papers in this section include the use of Neural Networks to analyze various types of data. Others cover various types of intrusion detection system, ranging from network intrusion detection systems to log and host-based systems as well as systems that correlate all these types of data.

The third section is to do with Neural Networks; it contains papers, books and articles referring to various types of neural network. Most of the material in this section covers recurrent Neural Networks.

The fourth and final section covers material on networks particularly TCP/IP networks and HTTP the protocol.

10.2.1 Security Exploits

[se0] Phrack 49 Oo Vol 7 Issue 49 Smashing the Stack for fun and profit
This is a tutorial on writing buffer overflow exploits by "smashing the stack". Very good reference for attack signatures of buffer overflows.

[se1] Mudge: How to write Buffer Overflows Tutorial
Self expalanator, paper on how to find and write buffer overflow exploits.

[se2] scutt/team teso, Exploiting Format String vunerabilities.
This is an excellent tutorial about a fairly recent type of vulnerability concerning printf and its relatives.

[se3] rain.forest.puppy, Perl CGI problems, Phrack Issue 55

Explains some exploits for perl program;

- *The use of \0 is different from c. (end of line in c). Therefore allowing you to enter a string in which is say root\0garbabe, the perl scanner may say is it "root" and of course it is not! The c program this is passed to will simply read it as "root" and the security check has been bi-passed.*
- *The use of directory traversals, and how to escape them etc. A program may use directory traversal in order to jump out of the standard web root directory into the main part of the system, this allows the user to execute arbitrary commands, the directory must be executable and the commands the attacker wishes to execute must be chmod'ed correctly.*
- *Use of pipe to get perls' open() to execute a program, i.e. open("/bin/sh|"). This allows a user to execute arbitrary commands, if it is known that the open call is used, and a cgi parameter is taken in representing some file, a pipe command will open the file as a command.*

[se4] Gregory Gilliss, CGI Security Holes, Phrack Magazine,Volume Seven, Issue Forty-Nine

Explains some more cgi exploits;

- *A bad choice of functions to use i.e. System calls provides possible security breaches.*
- *Redirects > in scripts allow the redirecting of the standard output causing files to be read.*
- *The use of null , again in perl provide a potential for poorly scanned input data to execute malicious code.*
- *Also outlines people grabbing password files.*

[se5] rain.forest.puppy, NT Web Technology Vulnerabilities, Phrack Magazine, Volume 8, Issue 54 Dec 25th, 1998

This paper iterates some interesting vunerabilities in Microsoft's IIS Web Server.

These do apply to any out of the box setup.

- *Standard scripts can be a problem. If the site is using a standard scripts and a hole has been discovered in this script, the attacker potentially has access to the source code of the script and can look for security vulnerabilities in it.*
- *IIS contained a security threat where it was possible to look in say c:\winnt\system32 or whatever, on any IIS Web Server.*
- *Using directory traversals. ../ has been a classic for IIS; many such vulnerabilities exist.*
- *Executing SQL Query's through IIS was a bug which allows an attacker to issue any queries to the database.[se6] Scambray Joel, McClure Stuart and Kurtz George: Hacking Exposed*

This book is a catalogue of exploits classifying them into various types of attack. It is quite useful for an overview of the area but contains little technical information with which to work with.

[se7] horizon jmcdonal@unf.edu, Defeating Sniffers and Intrusion Detection Systems,Phrack Magazine Volume 8, Issue 54 Dec 25th, 1998, article 10 of 12

This paper talks about various ways to defeat Sniffers and IDS's;

- *Fragmentation of TCP segment headers. This means, if the IDS is not capable of fragment reassembly packets, malicious packets can pass through fine.*
- *Bad stack decoding/cost extra options in IP header. Most systems use a naïve approach to stack decoding, as most IP packets don't contain things like options. If this is the case putting options in may lead the IDS to misinterpret the packet.*
- *Invalid sequence numbers. This can cause the IDS to crash if it doesn't contain a robust enough stack decoder.*
- *Many systems can launch off on wrong track, once spawned they will look at the TCP Session leaving malicious packets to sail through.*
- *De-Synchronization of TCP sequence numbers can cause the IDS to become confused.*
- *TCP checksum insertion. Invalid checksums can cause the IDS to slow down and possibly allow packets through.*

[se8] HTTP://www.theregister.co.uk/content/archive/22010.html

Top Vulnerabilities at the moment;

- *Default installs of operating systems and applications.*
- *Accounts with No Passwords or Weak Passwords.*
- *Non-existent or Incomplete Backups.*
- *Large number of open ports.*
- *Not filtering packets for correct incoming and outgoing addresses.*
- *Non-existent or incomplete logging.*
- *Vulnerable CGI Programs*

Top Windows vulnerabilities

- *Unicode Vulnerability (Web Server folder traversal).*
- *ISAPI extension buffer overflows.*
- *IIS RDS exploit (Microsoft Remote Data Services).*
- *NETBIOS - unprotected Windows networking shares.*
- *Information leakage via null session connections.*
- *Weak hashing in SAM (LAN Manager hash).*

Top Unix system vulnerabilities

- *Buffer overflows in RPC services.*
- *Sendmail vulnerabilities.*
- *BIND weaknesses.*
- *Remote commands.*
- *LPD (remote print protocol daemon).*
- *sadmind and mountd.*
- *Default Simple Network Management Protocol (SNMP) strings.*

10.2.2 Intrusion Detection Systems

[ids0] Helmer Guy G, Wong Johny S.K, Honavar Vasant and Miller Les. Intelligent Agents for Intrusion Detection. Ames, Iowa.
Interesting idea, taking a number of distributed IDS's and using them correlate the data together, to get a better model of the usage of the system. Uses System call traces, instead of live TCP/IP analysis.

[ids1] Cannady James, Artificial Neural Networks for Misuse Detection.
This paper iterates the issue of identifying the attack accurately and quickly. It compares Expert System approaches against Neural Networks.

Expert Systems - identifies exact matches by following list of rules and determining whether an attack is being attempted or not.
Disadvantages include:

- *Must add in attacks as they are discovered.*

The use of Neural Networks identifies a probability of the attack matching a previously seen pattern, therefore the quality of the network is based on how much it has seen previously.

Disadvantages of this approach include;

- *The system is a Black Box. It is difficult to know what it is doing.*
- *Attack traces are difficult to obtain.*

The paper discusses Misuse detection, again re-iterating the fact that a sequence of events warrants an intrusion. It re-iterates the fact that once an exploit is found, many others will use it giving a motivation for using misuse detection.

The paper gave some good ideas for collecting training data. Namely ISS Internet Scanner, Satan. It also gives a way of encoding the Data into the Neural Net which is worth reading when developing such a system.

References Worth Chasing:

[2] → Computer Vision Material[6] → International joint conference on Neural Networks Recurrent Networks[7] → Computers and Security Vol 13 No 6 pp 495-507[8] → An Intrusion Detection Model IEEE Transactions on Software Engineering, Vol SE-13 No 2.[9] → TISC[14] QStat [25][26]

[ids2]Balasubramaniyan Sundar Jai, Garcia-Fernandez Omar Jose, Isacoff David, Spafford Eugene, Zamboni Diego, An Architecture for Intrusion Detection using Autonomous Agents.
The paper gives a definition of Intrusion Detection;

"Problem of identifying individuals who are using a computer system without authorization, and those who are abusing their rights on the system ("escalation of priveledges."

The paper goes on to state that Intrusion Detection is not prevention, and provides some desirable characteristics of an IDS;

- *Always running*
- *Fault tolerant*
- *Resist Subversion (Attacks on IDS)*
- *Minimal overhead*
- *High Scalability*
- *Degradation of service should not affect the rest of the network.*
- *Dynamic configuration*

The paper goes on to review the use a Distributed IDS versus a Monolithic [15] approach. The problems which were identified with centralized system include;

- *One point of failure.*
- *Scalability problem, what happens when we want to increase the system to run on a bunch of machines.*
- *Analysis of network data can be flawed.*
- *Such a system is vulnerable to Insertion/Evasion attacks.*

The paper gave a good reference to a paper on Genetic Programming for Intrusion Detection [3].

Other issues which they consider in the system;

- *Portability*
- *Scalability*
- *Security*
- *Performance*

[ids3] Ghosh Anup K, Schwartzbard Aaron and Schatz Michael, Learning Program Behavior profiles for Intrusion Detection.

The system described in the paper is part of the DARPA Intrusion detection evaluation program. DARPA provide such systems with data for training and testing their systems.

They talk about Anomaly Detection and Misuse Detection, Misuse Detection tends to give low false positive and high false negative (many attacks get through, but ones that don't are identified correctly). Anomaly detection tends to give high false positive rates, has an inability to identify a particular attack. Must make sure training data is clean.

Strace – program that logs system calls was used in their investigation therefore it is a host based system. They iterate that Deterministic Finite State Automatons constructed by hand state result in state explosion these problems are described in [6]. They back up their reasoning of using system call traces by stating;

"A program can only abuse a system by making system calls."

They also talk about the time aspect of an attacker;

"Anomalous behavior tends to come in clusters."

The paper discusses the use of Back Prop examples in [7,1]. They then talk about recurrent Neural Nets such and implement the Leaky Bucket algorithm, used in temporal locality detection, and Elman networks- recurrent nets (look into these further).

[ids4] Ghosh Anup K and Schwartzbard Aaron, A Study using Neural Networks for Anomaly and Misuse Detection.

This is another DARPA project using their evaluation data again, the work is from Lincoln Labs at MIT. The paper iterates issue of being able to detect future unseen behavior rather than previously known attacks. The paper emphasizes the importance of low false alarm rates; high false alarm rates make IDS's impractical due to the amount of work the system admin must do.

The paper talks about the importance of being able to generalize over the input, i.e.

- *Detecting future normal behavior.*
- *Detecting future abusive behavior.*

The paper discusses the first proposal of Anomaly Detection, by Anderson in 1980- check out. It also discusses Expert System Anomaly detection models in [15].

The paper talks about a number of classes of attacks and informs the reader on the performance of the system with regard to a number of classes these include;

- *DOS*
- *Probing /Surveillance*
- *Remote to local*
- *User to root*

They also talk about the distinction between Network / Host based systems and draw conclusions as to their uses. The paper also interestingly talks about the human immune system [6] and its application to this topic, distinguishing between self and non-self.

They include a discussion of the anomaly detection model they use;

1. *Initialize NN with random data.*
2. *Train with normal data, so one area of the input space becomes identified with normal data. Temporal Locality problem again, leaky bucket algorithm is used.*
3. *ROC Curves Receiver Operating Characteristics. This seems to be industry standard for showing IDS results.*

The paper concludes by talking about their results, the misuse fails to generalize well and gives high false positives.

[ids5] Ranum J Marcus, LangField Kent, Stolarchuk Mike, Sienkiewicz Mark, Lambeth Andrew, Wall Eric, Implementing a Generalized tool for network monitoring.

This system is essentially a statistics gathering engine, shows a programming language for pulling packets apart and recording information or dropping them. Mentions libpcap, a useful interface as it exists on all UNIX systems and can be found for windows. The paper outlines the risk of buffering a loss of packets.

[ids6] Paxton Vern Bro: A System for Detecting Network Intruders in Real-Time

This is an intrusion detection language similar to [ids5]. However it is more advanced, it is based on rules that are coded using the language Bro, this also uses libpcap.

[ids7] Cannady James, Next Generation Intrusion Detection: Autonomous Reinforcement Learning of Network Attacks.

This paper outlines an interesting approach to detecting intrusions, very different from anything else seen so far. It uses an unsupervised Neural Network. This system works only with DOS attacks however.

The paper describes the use of CMAC Neural Networks. These could be useful and are worth investigating.

The system uses network de-gradation to train the system dynamically to detect intrusion.

[ids8] Schuba Christoph L, Krsul Ivan V, Kuhn Marcus G, Spafford Eugene H, Sundaram Aurobindom, Diego Zamboni: Analysis of Denial of Service Attack on TCP.

The paper involves a detailed explanation of the SYN flood attack, and ways to detect and resolve the threat. The presented solution involves basically by returning fake ACK's and waiting for the real ones later, to move connections to full open state.

[ids9] Machine Learning Techniques for the Domain of Anomaly Detection for Computer Security

This document involves a detailed discussion of AI techniques for anomaly detection, useful for ideas on types of machine learning. It talks about rule based /statistical models as well as the adaptive systems approach.

[ids10] Northcutt Stephen, Cooper Mark, Fearnow Matt, Karen Fredrick. Intrusion Detection Signatures and Analysis. Chpt 10-11

This book takes you through a number of log file formats including tcpdump output, SNORT output etc. It talks you through the kinds of things to look for in looking for attack signatures

Attacks of interest include;

- *DOS attacks – resource starvation versus bandwidth consumption.*
- *Land Attack- exploit code at* ftp.technotronic.com.
- *TOS bit was set to 0x30 for UDP. (Abnormal packet).*
- *Winnuke – URG to netbios port. This causes the netbios to expect data. The data never comes and RST packet is sent. Causing "blue screen of death".*

[ids11] 0x0b[0x10], A STRICT ANOMOLY DETECTION MODEL FOR IDS, Volume 0xa Issue 0x 05.01.2000 - P H R A C K M A G A Z I N E –

This paper describes a model of anomaly detection. Phrack people don't seem to like the author! Doesn't really give any relevant information.

[ids12] The Base-Rate Falacy and its implications for the difficulty of Intrusion Setection

This paper outlines an important point that the false alarm rate must be tiny for the system to be of any use. Uses Bayes theorem to show how ineffective a 99% accurate system is.

10.2.3 Neural Network material

[nn0] Bishop Christopher M: Neural Networks for pattern recognition Chpt 8.

This book is the definitive book on Neural Networks.

Chapter 8 gives an enlightening discussion of pre and post processing of data when using ANN's.

Points that were useful included;

- *Fewer inputs mean less dimensions, and therefore higher degree of generalization, and the system will train faster.*
- Pre-processing of data means less information and decreases dimensionality and so the above point applies.
- Re-scaling of input parameters so they are of similar range, makes the Neural Network more likely to work quicker and be more accurate.

Chpt 4. Gives a detailed discussion of MLP's

This chapter contains the necessary literature to understand and implement back propagation.

Page 267.

This page provides a discussion of the use of a momentum term in Back Propagation algorithm.

[nn1] Jeffrey Elman, Finding Structure in Time, 1990.

This paper gives an interesting proposal for a Neural Network architecture capable of finding structure in time. He goes through a number of ways to represent time in Neural Networks these include;

1. *Explicit serial order with dimensionality of inputs.*

The problems with this approach include;

- *input buffer.*
- *When should contents be examined.*
- *Rigid enforcement of duration.*
- *Does not distinguish between relational/absolute outputs.*

Elman proposes the use of an extra layer of "context units". These units are designed to capture internal state at each activation of the network. At each pass internal state is saved, thus representing temporal properties of sequential inputs. He concludes by discussing the change in the structure of the problem when represented temporally.

[nn2] Danilo P. Mandic, Jonathon A. Chambers. Recurrent Neural Networks for prediction. Learning Algorithms, Architectures and stability chpt 5.

This book discusses Recurrent Neural Networks and provides a number of architectures. It contains a discussion of the difference between feed forward vs Recurrent Neural Nets and indicates where both such topologies are useful;

- *Recurrent Neural Nets suffer from instability and are sensitive to noise*
- *Feed Forward Neural Nets may not be powerful enough to capture the dynamics of the problem.*

It outlines *how the Dynamics and/ complexity of a problem is implementation dependant and therefore each problem requires a different Neural Net.*

Recurrent Neural Network Architectures discussed in the book include;

Activation feedback
Output feedback
Locally Recurrent Globally feedforward category.
Elman/Jorden – limited in storing past information.
William Zipster Network— captures more info.

[nn3] Principe Jose C, Euliano Neil R, Lefebvre W Curt. Neural and Adaptive Systems Chapter 6, Chapter 3;

This book gives a good discussion and the necessary technical detail to implement Unsupervised Neural Networks. In particular Hebbian learning, Oja/Sangers rule and their use for PCA.

[nn4] Russel Stuart, Norvig Peter. Artificial Intelligence A Modern Approach.
This book provides the Back Propagation equations necessary to implement a Back Propagation algorithm.

[nn5] Ellis David. Non-Symbolic AI assignment; Training Algorithms for MLP networks.

This document compares a genetic algorithm with standard Back Propagation and Back Propagation with momentum.
[nn6] Recurrent Networks [1,2,3], *http://www.williamette.edu/~gorr/classes/cs449/rnn1.html* up on 22/12/01
Discussion of Recurrent Networks, back prop through time and patterns in time.

[nn7] Haselsteiner Ernst, What Elman Networks Cannot Do.

This paper shows a group of tasks which Elman networks are incapable of learning.

10.2.4 Computer Networks and Services

[n0] Stevens W. Richard: TCP/IP Illustrated Volume 1, The Protocols, Volume 2 The implementation, Volume 3 TCP for transactions, HTTP, NNTP and UNIX domain protocols.
These books provide an in depth guide to TCP/IP protocols, very useful for working out how TCP/IP works and some useful tricks in the implementation of TCP.

[n1] Packet Capture with libpcap and other low level network tricks.
This is a great tutorial showing you how to write a packet capture engine, using libpcap in c.

[n2] W. Richard Stevens. TCP/IP Illustrated Volume 3. Chapters 12/13 (HTTP Protocol/Packets found on an HTTP server).
This is a great book focusing on application level protocols. Statistics for internet services usage were reported outlining the growth of the HTTP protocol;
ftp://ftp.merit.edu/statistics - nfs back bone closed in 1995.

Talks about HTTP/1.0;
- *Multiple connections.(not persistant HTTP/1.0)*
- *Congestion avoidance and rtt etc. not passed on to all connections.*
- *Vunerable to lots of open half connections*

The Concept of a "Session" in HTTP was discussed and mentions;

(Kwan McGrath and Reed 1995. User Access Patterns to NCSA's World Wide Web server.)

The book outlines the problem with one doc per connection, it reacts badly with TCP as slow start and all the other things are not shared between connections. Keeping connection open if cache size of response is known was also discussed.

[n3] RFC HTTP/1.1.
The Request For Comment on HTTP/1.1 protocol. Main points that were found useful;

- *Connections now persistent by default. Before one connection was made for every request.(Memory/cpu time saved for Clients/servers/gateways/proxies/routers).*
- *Pipe-lined requests possible. This allows multiple connections to open from the same client making a number of requests. (Congestion reduced.)*
- *Latency better, as there are less half open connections about.*
- *HTTP's evaluation better is better.*
- *Connection header field, explicitly close connection. HTTP/1.0 assumed the connection will close, HTTP/1.1 says user must explicitly close it.*

The RFC states includes the grammar for HTTP which was used in the implementation of WebIDS.

[n4] http://www.mit.edu/people/mkgray/net/web-growth-summary.html

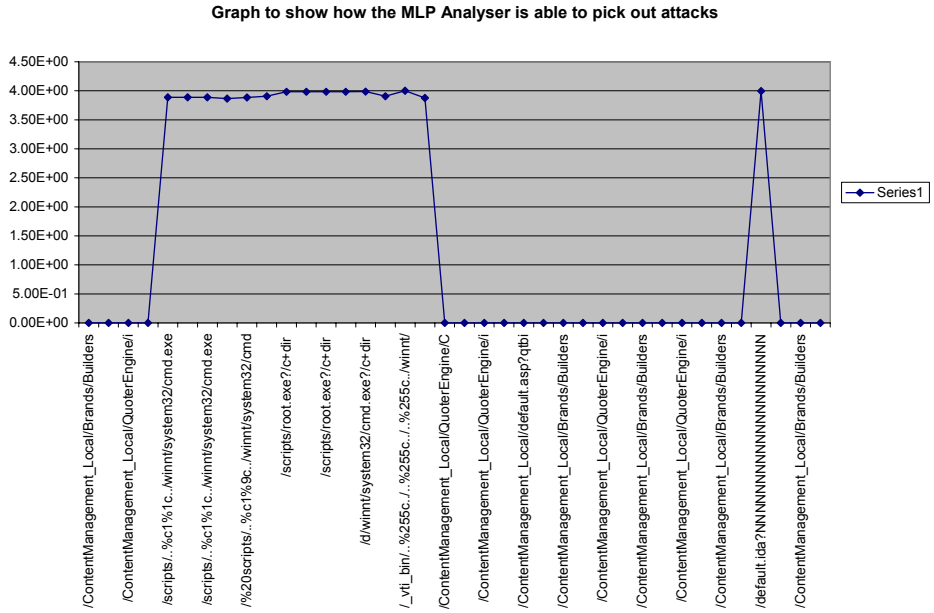
This site gives out statistics on what internet services people are using. It is very useful for determining the most popular serv

10.3 Excess Test data

10.3.1 Selected URLS and their classifications via the MLP network

```
/ContentManagement_Local/QuoterEngine/inc/stdlib.js  2.16E-05
/ContentManagement_Local/QuoterEngine/inc/validate3.js      5.90E-06
/ContentManagement_Local/default.asp?qtbid=5&qtcid=QL&qtqecid=63&qtspecid=&qtspecqmid=
        3.56E-05
/ContentManagement_Local/Brands/BuildersDirect/scripts/GeneralUtility.js   1.80E-05
/ContentManagement_Local/QuoterEngine/inc/stdlib.js  2.16E-05
/ContentManagement_Local/QuoterEngine/inc/validate3.js      5.90E-06
/ContentManagement_Local/default.asp?qtbid=5&qtcid=QL&qtqecid=63&qtWizardPage=37&qtspecqmid=
        4.02E-05
/ContentManagement_Local/Brands/BuildersDirect/scripts/GeneralUtility.js   1.80E-05
/ContentManagement_Local/QuoterEngine/inc/stdlib.js  2.16E-05
/ContentManagement_Local/QuoterEngine/inc/validate3.js      5.90E-06
/ContentManagement_Local/Brands/BuildersDirect/scripts/GeneralUtility.js   1.80E-05
/ContentManagement_Local/QuoterEngine/inc/stdlib.js  2.16E-05
/ContentManagement_Local/QuoterEngine/inc/validate3.js      5.90E-06
/ContentManagement_Local/QuoterEngine/Images/16x16Note.gif  4.66E-05
/scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir   3.889031336
/scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir   3.889031336
/scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir   3.889031336
/scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir   3.865881341
/%20scripts/..%c1%9c../winnt/system32/cmd.exe?/c+dir        3.887250169
/scripts/..%25%35%63../winnt/system32/cmd.exe?/c+dir        3.906648224
/scripts/root.exe?/c+dir      3.983254482
/scripts/root.exe?/c+dir      3.983254482
/scripts/root.exe?/c+dir      3.983254482
/c/winnt/system32/cmd.exe?/c+dir      3.984855172
/d/winnt/system32/cmd.exe?/c+dir      3.987073659
/scripts/..%255c../winnt/system32/cmd.exe?/c+dir    3.905079296
/_vti_bin/..%255c../..%255c../..%255c../winnt/system32/cmd.exe?/c+dir      3.999631297
/msadc/..%255c../..%255c../..%255c/..%c1%1c../..%c1%1c../..%c1%1c../winnt/system32/cmd.exe?/c+
dir    3.877184856
/ContentManagement_Local/QuoterEngine/Content/GlossaryPopup.asp?Term=Paint        6.61E-06
/ContentManagement_Local/QuoterEngine/inc/stdlib.js  2.16E-05
/ContentManagement_Local/QuoterEngine/inc/stdlib.js  2.16E-05
/ContentManagement_Local/QuoterEngine/inc/validate3.js      5.90E-06
/ContentManagement_Local/default.asp?qtbid=5&qtcid=QL&qtqecid=23    9.04E-05
/ContentManagement_Local/default.asp?qtbid=5&qtcid=QL       1.53E-04
/ContentManagement_Local/Brands/BuildersDirect/scripts/GeneralUtility.js   1.80E-05
/ContentManagement_Local/QuoterEngine/inc/stdlib.js  2.16E-05
/ContentManagement_Local/QuoterEngine/inc/validate3.js      5.90E-06
/ContentManagement_Local/default.asp?qtbid=5&qtcid=QL&qtqecid=23    9.04E-05
/ContentManagement_Local/Brands/BuildersDirect/scripts/GeneralUtility.js   1.80E-05
/ContentManagement_Local/QuoterEngine/inc/stdlib.js  2.16E-05
/ContentManagement_Local/QuoterEngine/inc/validate3.js      5.90E-06
/ContentManagement_Local/default.asp?qtbid=5&qtcid=QL&qtqecid=64    5.77E-05
/ContentManagement_Local/Brands/BuildersDirect/scripts/GeneralUtility.js   1.80E-05
/ContentManagement_Local/QuoterEngine/inc/stdlib.js  2.16E-05
/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN%25u9090%25u6858%25ucbd3%25u7801%25u9090%25u6
858%25ucbd3%25u7801%25u9090%25u6858%25ucbd3%25u7801%25u9090%25u9090%25u8190%25u00c3%25u0003%25
u8b00%25u531b%25u53ff%25u0078%25u0000%25u00=a       3.995857781
/ContentManagement_Local/default.asp?qtbid=5&qtcid=QL       1.53E-04
```

10.3.2 Graph to show exploits being detected



We can see the ease at which the MLP is able to pick out the Code Red and Nimbda attacks.

10.4 Glossary

Anomaly Detection Model	Where a system is trained with normal data, following this it is able to detect deviations from the normal.
Attack	Attempt to misuse or compromise a computer system in any way.
Attacker	An attacker is a person or program that has purposely set out to misuse a computer system in some way.
CGI	Common Gateway Interface, The interface specification defining how information can be passed back and forth from browser to client.
Correctness	The percentage of correctly identified attacks the system has made
Data sets	The data used to train a neural network, usually labelled with the correct classification.
False Negatives	The number of anomalous sessions incorrectly identified as normal sessions.
False Positives	The number of normal sessions incorrectly identified as anomalous sessions.
Firewall	A Program or piece of hardware which filters packets before forwarding them onto a host.
Fragmentation	Where packets are broken up and sent in separate IP packets, the application must re-assemble the packet on arrival. Note TCP headers may be fragmented.
Generalisation	The ability to generalise input over passed seen behaviour. For example if you see one cat you can then identify any cat you see in the future as a cat.
HTTP	Hyper Text Transfer Protocol - The protocol of the World Wide Web.
HTTP message	The message sent back and forward from HTTP server to client.
HTTP request	The HTTP message sent to a web server asking for a particular resource from the server, such as a web page.
HTTP Session	The collection of HTTP messages exchanged between server and client during one clients active session.
Hyper-plane	a plane of seperation through data points in d-dimensional space, a 3d hyperplane is a plane, a 2d hyperplane is a line.
IDS	Intrusion Detection System - System used to detect intruders to a computer system
IP Spoofing	Where an IP packet appears to have been dispatched from a different host from the one it was actually dispatched from.
javacc	The java compiler compiler - given a grammar in ebnf style notation will generate an appropriate parser
JPCap	A packet capture library for Java.
Linearly Separable	Where there exists a single hyper-plane of seperation between data points in d dimensional space.
Misuse detection model	Where a system is trained with a number of instances of misuse, the system will then detect whether new data is an instance of misuse, by determing its similarity to the instances of misuse.
Packet Sniffer	A program which intercepts data packets on a computer network.
Promiscuous	A class of learning algorithm where the weights are adjusted according the known ideal outcomes of the input sequence.
Proxy server	A server that makes connections to other hosts on behalf of a client.
Supervised Learning	A form of learning for Neural Networks, where the Neural Network is adjusted precisely due to the error of its outputs.
Tcpdump	Tcpdump is a popular Unix program for dumping network traffic.
The Service	The same as “The System”
The System	The product that is being developed, i.e. an intrusion detection system using the anomaly based model with a neural network analysis engine.
Weight matrix	In neural networks, the internal set of weights, which describe the input/output mapping of the network.

10.5 User Manual

10.5.1 Installation

Installation of the system requires the following system capabilities;

- Java Software Development Kit Version 2 or above.
- The libpcap for UNIX or winpcap for Windows environment.
- The JPCap library – the libpcap interface to java.
- A 486 or above preferably running Linux, MS Windows is ok.
- An Ethernet Card connected to an Ethernet Network.
- If the system is to be deployed as part of a firewall, a second Ethernet Card is required.

To install the system then follow these simple instructions;

1. Set up your java system (follow instructions which come with java).
2. Set up libpcap/winpcap (follow instruction which come with these).
3. Unzip the file webids.zip into the directory you wish to deploy the system. We recommend /usr/sbin/. Make sure the classpath has been set to this directory.
4. The system is now ready for training.

10.5.2 Configuring using the WiComponentFactory class

The WiComponentFactory class contains a number of methods which return the correct objects for the rest of the system. A list of the available EvReaders and AeAnalysers are available for you to choose;

In the method makeAnalyser the following analysers can be used (AeAnalyserDump) is used here.

```
// uncomment if you want an MLP Neural Network analyser
//return aemlp;
// uncomment if you want the PCA Neural Network
//return aepca;
// uncomment if you want the MLP with PCA pre-processor
//return aepcamlp;
// uncomment if you would like the group occurence analyser.
//return aegc;
// uncomment if you want the dump analyser.
return aedump;
```

In the method makeReader the following readers can be used;

```
// uncomment if you wish to use a TCP/IP socket
return  new EvReaderSocket();
// uncomment if you want to read from text files.
//return new EvReaderText();
// uncomment if you want to read raw packets from the ethernet card.
//return new EvReaderJPCap();
```

10.5.3 Collecting the training data

To collect the training data, you must first configure the system to use the correct EvReader and AeAnalyser. Edit the WiComponentFactory class by un-commenting the AeAnalyserDump analyser. The EvReader class you use depends on where the data is to be collected from.

By default the system dumps the HTTPRequest strings into the file aedump.log. This may be changed by adding the filename where you wish to dump the requests to in the constructor of the object;

```
AeAnalyser aedump = new AeAnalyserDump(filename);
```

To begin collecting the data, at the prompt type;

tsunx%java WebIDS.WiOpenLive

This will begin capturing the data and writing to the specified file and to the console. The program will run until the original source you are reading from returns or you type ^C.

10.5.4 Training

1. The training phase is easy, first open up WiComponentFactory and select the analysers and readers you wish to use.
2. Then create a directory by the name of the site you want to train, e.g. if you want to train the system on a site called trends.com we would call the directory Trends.
3. Then create two sub directories, one called normal and one called anomalous.
4. Copy the trace data you have collected into the normal and anomalous sub directories as required.
5. Then at the prompt type; tsunx%java.WebIDS.TrainEnv

This will provide statistics for later analysis and drop them to various files. Look in the source of the individual analysers to discover the format of the statistics produced.

10.5.5 Testing

Tests can be run as though the system is in a live state, except a set of marked dump files should be used. This will generate a load of logs, as though the system was being attacked which you can analyse and determine the performance of the system.

10.5.6 Deploying

1. The system can be deployed by simply configuring the WiComponentFactory class.
2. Then type at the prompt; tsunx%java WebIDS.OpenLive